AD-A110 858 PRC GOVERNMENT INFORMATION SYSTEMS MCLEAN VA F/G 9/2
IMPROVED MICROPROCESSOR DESIGN.(U)
OCT 81 L E KLET, R E CRANDALL, J H GLENDE F30602-80-C-0171
UNCLASSIFIED PRC-R-3412 RADC-TR-81-273 NL

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-81-273	2. GOVT ACCESSION NO.: AD-A110 858
	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): IMPROVED MICROPROCESSOR DESIGN	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report May 80 - May 81
	6. PERFORMING ORG. REPORT NUMBER: PRC-R-3412
7. AUTHOR(s): L. E. Klet, M. A. Okulski, R. E. Crandall, J. E. Simpson, J. H. Glende	8. CONTRACT OR GRANT NUMBER(s): F30602-80-C-0171
9. PERFORMING ORGANIZATION NAME AND ADDRESS: PRC Government Information ~~Sciences~~ 7600 Old Springhouse Road McLean VA 22102	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 45941640 62702F
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (IRDA) Griffiss AFB NY 13441	12. REPORT DATE: October 1981
	13. NUMBER OF PAGES: 192
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Lisle Sanborn (IRDA)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Networking	Bandwidth	Bus Controller
Communications	Firmware	Inter-bus Link
Microprocessor	Bus Cycle	Computer Systems
Exchange bus	Direct Memory Access	Distributed Control
Distributed Architecture	Intelligence Data Processing	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This Final Technical Report is submitted to Rome Air Development Center in accordance with data item A005 of contract F30602-80-C-0171. The scope of this report includes engineering services provided under the above referenced contract between May 1980 and May 1981 by PRC/GIS. The intent of this approach is to convey technical aspects and applicability of the design for the improved microprocessor.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

i

Nav 4127-1

This report details the technical improvements proposed to provide an improved Micro Programmable Controller (MPC). The MPC is an innovative combination of hardware and software, synergistically coupled, to produce a truly distributed computer system. The hardware consists of an array of micro-computer subsystems called ports which communicate with each other via a high-speed, bidirectional, asynchronous exchange bus. These ports are capable of communicating externally via an I/O hardware interface which supports various serial protocols, such as the DDCMP.

The objective of these technical design improvements was to enhance the performance of the Microprogrammable Controller (MPC) without altering its basic architectural characteristics by redefining the implementation of certain logical control elements and by migrating to hardware some control features currently accomplished in firmware. The Bus Interface Electronics and Bus Controller were redesigned to provide greater bus and port-to-port bandwidth.

The Inter-Bus Link (IBL) was designed to efficiently transfer bus cycles between physically isolated exchanged busses (SSBUS). Finally, the existing MPC system-level firmware was redesigned to operate the improved MPC.

PREFACE

This document is the final technical report listed as data item A005 of the Contract Data Requirements List for Rome Air Development Center (RADC) contract F30602-80-C-0171. Work described herein was performed by Planning Research Corporation (PRC), McLean, Virginia, in support of Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, New York, during the period May 1980 through May 1981.

RADC project engineer was Mr. Lisle Sanborn, Intelligence Systems Section, Intelligence Data Handling Branch of the Intelligence and Reconnaissance Division.

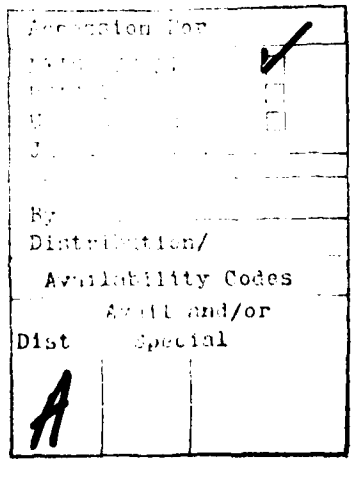


TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

LIST OF FIGURES

LIST OF TABLES

SECTION I
INTRODUCTION

This final technical report is submitted to Rome Air Development Center (RADC) in accordance with data item A005 of contract F30602-80-C 0171. The principle project reference relevant to this technical report is the Statement of Work (SOW) for the Improved Microprocessor Design contract, RADC PR NO. I-0-4344 dated May 7, 1980.

1.1 Purpose. The intent of this report is to provide the reader with an understanding of current Micro Programmable Controller (MPC) capabilities and the design of the Improved MPC resulting from engineering services provided under the referenced contract by PRC.

1.2 Scope. The scope of this report includes engineering services provided under the above referenced contract and improved MPC hardware/firmware characteristics as designed by PRC personnel. This approach conveys the current operational state of the MPC development and provides the reader with an appreciation of how these new capabilities could be utilized by other agenices throughout the Air Force and the Department of Defense.

1.3 Report Organization. This report is organized into five major sections as follows:

- Section 1 provides information concerning contract F30602-80-C-0171 and its associated SOW, RADC PR NO. I-0-4344; statements of purpose and scope; and concludes with report organization.

- Section 2, background data providing the history of the MPC, and the requirements for an Improved MPC.

- Section 3 presents each major component of the MPC by describing hardware, communications (firmware) and software architectures that collectively make up the current MPC.

- Section 4 provides design details for the Improved MPC in satisfaction of each requirement under this contract.

- Section 5 provides a summarization of the specific design details and the implications of the new design. This section is followed by an appendix containing a list of hardware and software acronyms and terms used in this report.

SECTION II
SUMMARY OF REQUIREMENTS

The purpose of this section is to provide a brief background of the events that led to the development of the MPC, and an overview of the current architecture with possible improvements that could be made to it. This is followed by a summary of requirements for the improved MPC design as stated in the SOW.

2.1 Background. This subsection described the technical problems that led to definition of the MPC architecture and the history of the MPC development including its integration into the SAC Intelligence Data Handling System (IDHS) environment.

2.1.1 History of the MPC. Beginning in 1966, Rome Air Development Center (RADC) and Planning Research Corporation (PRC) personnel began designing the first large-scale, on-line intelligence system within the Department of Defense (DoD) at Headquarters Strategic Air Command (SAC). This system was named PACER or Program Assisted Console Evaluation and Review. PACER achieved an initial operating capability in November 1970 with on-line, real-time, data base update and applications support being provided by 16 Bunker Ramo graphics consoles (BR-90s) and 32 Radio Corporation of America (RCA) textual consoles. The BR-90 consoles were interfaced to the PACER Honeywell 6080 mainframe by two uniquely developed hardware components called Channel Control Units (CCUs) while all 32 RCA consoles were interfaced to the mainframe by a Digital Equipment Corporation (DEC) PDP-15 minicomputer. With this hardware configuration, failure of either CCU or the PDP-15 caused eight BR-90s or all 32 RCA consoles respectively to be unavailable for use. Furthermore, the supporting PACER system and console handling software could not handle additional consoles of either the same type or of a different type.

By 1975 changing requirements dictated a need for increased capabilities. Among those required were: increased system capacity and reliability, direct access to communications circuits, existing terminal replacement and the means to support increased numbers of terminals and multiple terminal-types, the ability for a single terminal to access multiple systems, and the ability to electrically intertie the Honeywell 6080s with five DEC PDP-11 minicomputers.

In short, PACER required a capability to satisfy the following system requirements:

- o Normalize the BR-90 and RCA Textual Console Interface
- o Standardize the PACER to console communication
- o Provide high reliability and availability
- o Incur minimum PACER executive and application software impact
- o Provide future system flexibility and extensibility to meet near-real-time information processing objectives

There were two standard systems approaches for providing the required PACER capabilities. The first was to upgrade the host processor; however, since PACER already was supported by the Honeywell top-of-the-line 6080 system, this alternative was not feasible. The second approach was to add one or more Front-End Processors (FEPs) to provide needed communications capabilities. With the large screen size of the BR-90 console and the high system interaction rates, it was determined that an FEP would have to handle a steady-state load of 30,000, and a peak load of 60,000, characters-per-second to provide adequate system response time to PACER users. Loads of this magnitude were above the effective range of available FEPs and would therefore place additional work on the Honeywell 6080 mainframe for network control. Additional PACER host workloads were considered undesirable.

During 1975, RADC engineering personnel were investigating the benefits that might accrue to USAF Commands from integration of microprocessor technology into various Command ADP environments. This work together with consultation between RADC engineers and PRC technical personnel led to definition of a concept using an array of asynchronously operating microprocessors and a high-speed communications bus to resolve certain PACER problems. This concept was later developed and implemented by PRC and called the Micro Programmable Controller (MPC). Under this concept, all PACER consoles were directly interfaced to the MPC and operationally implemented in August 1979. The MPC was integrated with the PACER Honeywell 6080s by new subexecutive software called PACER Communications Module (PCOM) which communications with the MPC in a message mode through the standard Honeywell Datanet 355 FNP. PCOM provides a standard communications interface with the Honeywell 6080s and performs required transliteration between the network and PACER character sets. It also performs necessary message data compression and expansion functions within the network.

Implementing the PACER MPC resulted in eliminating the two unique CCUs and the PDP-15 computer -- all critical network components -- and provided needed flexibility in switching individual BR-90s or RCA Textual Consoles between the Operational and Test PACER Honeywell 6080 computers.

Subsequent PACER executive software enhancements in concert with MPC developments removed the previous constraint of two console types and 48 total consoles. Today, the PACER system supports a complement of 67 dual-screen UNIVAC OJ-389(V)/G workstations, twenty local printers, and is interfaced with two PDP-11 based AN/GYQ-21(V) systems. Since its implementation, the MPC has had one system failure operating 23 hoursper-day, seven days-a-week. The system was restored to full operation in 30 minutes.

2.1.2 MPC Architectural Overview. This paragraph briefly describes the MPC design concept, the physical configuration, and system-level software that collectively embody the SAC MPC. This discussion is intended as an aid for the reader in relating the current MPC concept to subsequent requirements discussions for this specific contractual effort to improve the MPC hardware architecture.

The MPC is an innovative combination of microcomputer hardware and software synergistically coupled to produce a distributed computer system. Physically, the MPC is composed of an array of microcomputers called "ports" interconnected by a common high-speed communications bus. Each port is a complete computer consisting of a microprocessor, memory, and I/O interfaces with the bus and an optional attached device.

Each port within the MPC architecture operates independent of, and in parallel with, all other ports. Assuming a 50-port configuration using INTEL 8080A microprocessors, the MPC is capable of executing 25-million instructions per second -- considerably in excess of instruction execution rates for the largest mainframes available today with the exception of supercomputers.

The MPC architecture distributes dedicated processing resources (i.e., ports) as close as possible to a data source or device. This design ensures that network processing on behalf of one user device cannot diminish resources required by another user device. The design also reduces the processing burden of an attached mainframe by offloading some processing that it would otherwise have had to accomplish. The result of this approach when implemented within existing host-centered, saturated, architectures is frequently an improvement in system response time to user system requests since such requests can be satisfied entirely within the MPC without a burden on mainframe resources.

The MPC architecture is totally modular with respect to both hardware and software. Therefore, as new interfaces or processing requirements are identified they can be satisfied by addition of specially designed ports without affecting the existing MPC configuration. This architectural feature is the basis of MPC expandability, flexibility, and adaptability.

At the software level, the current MPC exchange bus or XBUS is controlled by the Inter-Port Communications (IPC subsystem, a copy of which resides in each port. The IPC software lies between the XBUS hardware and any port application software. Consequently, an IPC in one port is able to establish and conduct dialogues with an IPC in another port without assistance or awareness of any intermediary third port. The totally distributed nature of IPC allows any number of simultaneous two-port dialogues to occur across the XBUS within MPC hardware limits and communications requirements of the moment. Since IPC software lies between the XBUS hardware and any port application software, it provides hardware transparency for all other software subsystems executing within a port. This transparency is so complete that if the XBUS hardware were changed the only software impact would be upon IPC. The MPC architecture provides another level of hardware transparency in that all attached MPC devices communicate through a standard data protocol. It is the application level software within each port that is responsible for translating between the data level protocol of any attached device and the MPC standard data protocol. The hardware transparency features of IPC are what make the MPC architecture so flexible in resolving problems commonly found in today's large-system architectures.

The final major architectural feature of the MPC is embodied in the Inter-Bus Communications (IBC) subsystem. The IBC software acts as an extension of IPC to permit ports that do not physically reside on the same XBUS to conduct port-to-port dialogues. This feature provides network growth flexibility in that multiple MPC systems can be physically separate within a network architecture but logically and electrically interconnected.

2.2 Required MPC Design Improvements. The MPC presents an opportunity to address problems not currently solvable with traditional architectures through the application of parallel processing techniques. These techniques when applied to the large integrated problems found in IDH Systems require that those problems be functionally decomposed in order to apply separate processing resources to each. Once a problem has been decomposed and implemented on several processors, the need for data communications and process coordination becomes paramount. The MPC through its numerous microcomputers interconnected by a common bus managed by the Inter-Port Communications (IPC) firmware subsystem has solved the basic data communications and process coordination problem.

However, improvements in the existing MPC architecture remain to be developed in order for the communications and coordination functions to be accomplished at a speed substantially closer to that at which they are accomplished within the traditional mainframe. This is especially true since more data and process control communications must be accomplished in a functionally decomposed, parallel, implementation than in the serial mainframe approach. This inter-process communication is the price that must be paid to reduce processing time through the application of parallelism.

Improvements in the MPC architecture to decrease the time it takes to accomplish this inter-process communication can be made by increasing the bandwidth of the MPC in two major areas:

- Data communications bandwidth
- Process coordination bandwidth

Data communications bandwidth is a measurement of the amount of data that can be transferred between two ports during a given period of time. Data communications bandwidth can therefore be analyzed by determining the raw bandwidth of the XBUS (bus bandwidth) and the effective bandwidth at which two ports can exchange data (port-to-port bandwidth). XBUS bandwidth is a measurement of the bus cycle rate and the number of bits of data transferred

per bus cycle. In the current MPC implementation, the bus cycle rate is 2 megacycles per second. Because the XBUS addressing lines are shared (multiplexed through) with the XBUS data lines, 16 bits of data is transferred every other bus cycle. The effective bandwidth of the current MPC XBUS is therefore 16 megabits per second (16 bits of data X 2 million bus cycles per second/2 = 16 megabit per second). Port-to-port bandwidth is a measurement of the rate at which data can be transferred from port memory to the XBUS and the amount of data moved during each transfer. In the current MPC implementation, 16 bits of data is transferred from the port memory to the XBUS at 15.75 kilo cycle per second rate. The maximum port-to-port bandwidth in the current MPC implementation is therefore 250 kilo bit per second (16 bits of data X 15.75 thousand cycles per second = 250 kilobit per second).

Process coordination bandwidth is a product of the amount of time expended in the course of a dialogue to ensure synchronization of two ports, accomplish the transfer of data, ensure data integrity, and release each port from the dialogue. These functions are required in each IPC dialogue. The proportion of time required to complete overhead processing in relation to the time spent actually transferring data is too excessive in the current architecture. This problem is most evident in the case of control dialogues where only a small amount of data is transferred to facilitate the initiation of parallel tasks or update statuses. Assuming a data transfer of ten words (16 bits each), about 700 microseconds would be required to complete the port-to-port transfer. The overhead processing would require about an additional 1800 microseconds. The overall duration of the control dialogue then becomes approximately 2.5 milliseconds. Process coordination, with respect to the control dialogue, is then the limiting factor in the amount of parallel activities or tasks each MPC port can sustain in a given period of time. In the current architecture, this figure is 400 control dialogues per second. (1/2.5 milliseconds per control dialogue = 400 control dialogues per second).

In terms of MPC architectural characteristics, bus bandwidth can be improved by decreasing the amount of time between bus cycles, transferring more bits in parallel per bus cycle, or by bus segmentation. Bus sementation implies multiple busses linked together operating as a single system. Improved bus bandwidth also improves port-to-port bandwidth, however, additional improvements are possible. The main alternatives are to employ a faster microprocessor in the ports; use of parallel Direct Memory Access (DMA) techniques in transferring data across the bus, thus freeing the microprocessor to create the next transfer request; or both. Process coordination bandwidth is improved if bus and port-to-port bandwidths are improved, again however, additional improvements are possible. For example, portions of the firmware overhead associated with establishing contact with another port, lockon of one port to another to accomplish a dialogue, and the termination of that dialogue, could be migrated into the bus interface electronics. Firmware overhead processing would then execute asyncronously to the port processor. In addition to bandwidth, there are also reliability, flexibility, extensibility, and other engineering improvements that could be made.

The paragraphs below expand upon general requirements for an improved MPC hardware design as reflected in the Statement-of-Work (SOW) for this procurement.

2.2.1 Improved Bus Design. The objective of the improved bus design is to remove existing throughput limitations and increase performance without significantly altering the basic MPC architectural concept. The present bus bandwidth can be increased by demultiplexing the data and address lines and by minimizing the number of handshaking signals required for a bus cycle data transfer. Theoretically, an XBUS cycle of close to 200 nanoseconds should be achievable using these techniques.

2.2.2 Improved Bus Controller. The objective in improving the MPC bus controller is to increase bandwidth by redesigning the bus control signals. This design approach will be less complex than the current version due to related design changes being incorporated into the Bus Interface Electronics discussed in paragraph 2.2.3 below, and because of planned changes in the manner by which bus interface control signals will be generated. The improved bus controller will consist of one XBUS module that will control the arbitration signals for the physical bus.

2.2.3 Improved Bus Interface Electronics. This area of performance improvement addresses such issues as XBUS bandwidth, port-to-port bandwidth, DMA data transfers across the XBUS, and hardware vs. software port dialogue establishment. Currently, all XBUS cycles require direct initiation by the port processor; therefore the processor must be directly involved during all phases of dialogue establishment and data transfer. One objective in improving MPC performance involves decoupling of XBUS activities from other port processor functions. By migrating port dialogue establishment functions of IPC from firmware to hardware this element of delay or overhead will be greatly reduced, thereby decreasing overall dwell time of a dialogue. Additionally, by using hardware DMA techniques to affect data transfer the processor will be decoupled from the actual data transfer and be able to proceed at the cycle time of memory and/or the XBUS (whichever is the limiting factor).

2.2.4 Inter-Bus Link (IBL). The IBL design concept is an improvement to the existing MPC Inter-Bus Communications (IBC) port that permits multiple physical XBUSs to be electrically interfaced. The IBL design will be implemented in a new port capable of initiating and receiving bus cycles from all other ports on the same physical XBUS. The primary function of IBL will be to buffer data from a bus cycle on one physical XBUS to a second physical XBUS all within one logically extended MPC XBUS.

2.2.5 Improved Firmware. The objective of this task will be to redesign existing MPC system-level and interface-specific port firmware subsystem to support components of the improved MPC design. Specifically, the Inter-Port Communications (IPC) subsystem firmware that controls and communicates with the XBUS interface circuitry will be redesigned. This change is required because of the bus interface circuitry design change and some IPC functions previously accomplished in firmware will now be accomplished by hardware in the new design.

2.2.6 Reliability/Performance Requirements. Several techniques are being used in the improved MPC to provide greater reliability and performance. As mentioned in previous paragraphs much of the effort for this design is aimed towards increased performance. In addition, efforts are being expended to improve the overall reliability of data transfer in the MPC by such means as providing hardware checksum techniques and including error recovery in the bus interface unit.

2.2.7 Timing Requirement. Many of the new timing requirements are being met by changes in XBUS interface circuitry. Also, this implies some changes in the hardware interface portions of IPC.

SECTION III
CURRENT MPC DESCRIPTION

The current MPC can be logically divided into two major areas - MPC hardware and MPC software/firmware. In the following sections each of these topics will be separately addressed describing the MPC in terms of existing capabilities.

3.1 MPC Hardware Description. The MPC architecture is an array of microprocessors called ports, interconnected by a common, high speed exchange bus (XBUS). Each port is a complete computer consisting of a microprocessor, memory, and I/O interfaces to the XBUS and, in some cases, to an attached device. The XBUS allows the interconnection of ports on a demand basis by resolving conflicts and granting bus cycles on a rotating priority basis. Figure III-01 Bus/Port Relationship illustrates the basic architecture of the MPC. A more complete discussion of this hardware is presented below.

3.1.1 Cabinet. The MPC cabinet consists of a stand-alone enclosure containing the card cage in which the ports are mounted, the exchange bus (XBUS), one DEC 11/23 File Management System port, an optional floppy disk, power supply, and cable transition plate.

3.1.1.1 Enclosure. The enclosure is a modified commercially available 24" rack CABTRON cabinet. To meet EMI shielding requirements, the CABTRON cabinet is fitted with specially made doors, side panels, and back panel. Access is currently via the top for card insertion/removal and the front for I/O cabling connect/disconnect. The DC on/off controls and indicator lights are accessible from the front exterior. Commercially available foam and metal gaskets are used to provide necessary EMI shielding between removable panels/doors and the enclosure frame. If it is necessary to increase the

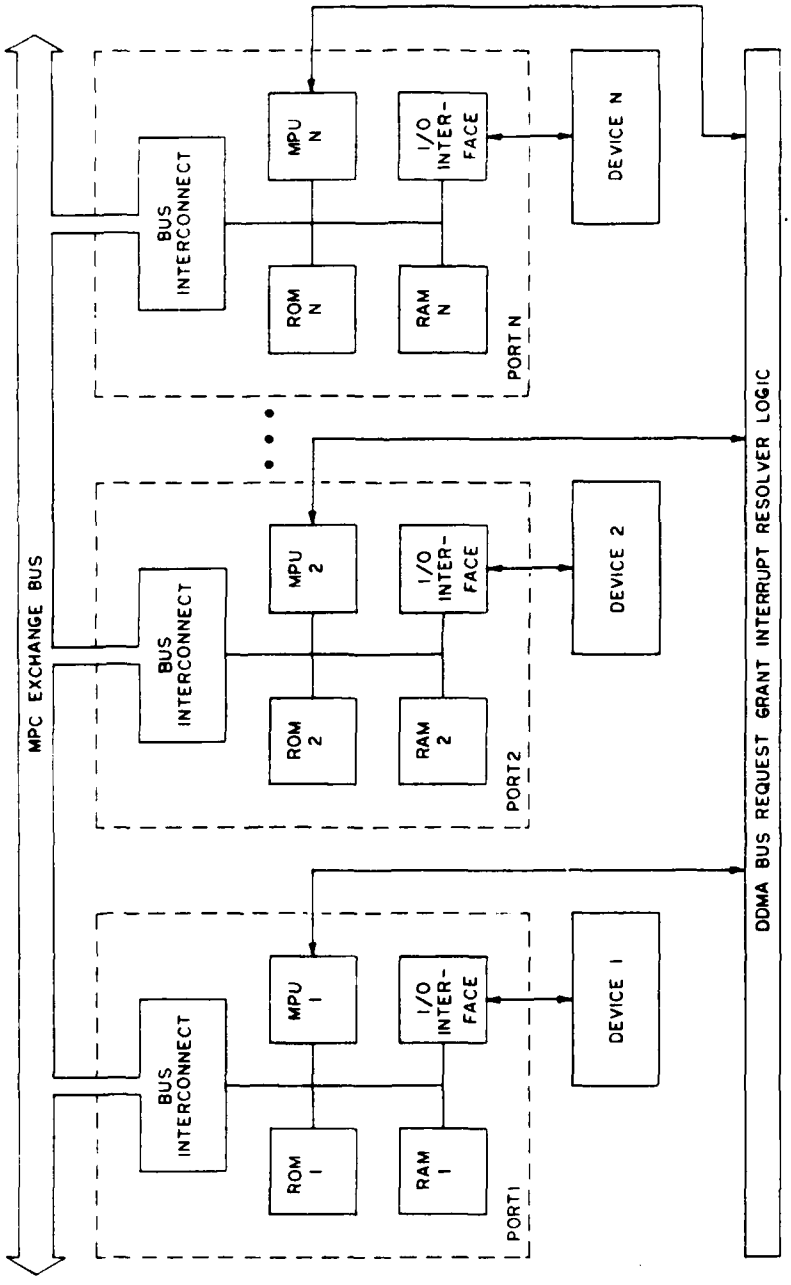


Figure III-01. BUS/Port Relationship

overall system beyond a single cabinet, the cabinets may be bolted together, side-by-side, with interconnecting cables running internal to the cabinets.

3.1.1.2 Card Cage. The card cage is mounted inside the cabinet enclosure via the 24" rack mount on the frame. The card cage is custom fabricated from 12 Ga steel to house up to 24 cards in pairs, with .75 inch spacing between each card. The XBUS motherboard at the rear of the cage is actually supported by the Zero Insertion Force (ZIF) connectors which are bolted to the cage. Presently, the rear of the cage consists of 2 rows of connectors. Only the bottom row is used by the XBUS motherboard with the top row available for future expansion. All of the connectors in the card cage are ZIF connectors manufactured by AMP. The connectors serve as both a card guide and electrical connection. The unique sequential feature of these connectors allow power to be applied prior to signal connection allowing card removal/-insertion with DC power supplied to the cage and while other cards remain functioning.

3.1.1.3 Optional Floppy Disk Space. The cabinet also has space for an optional 8" floppy disk with its own power supply. This floppy is directly below the card cage, enclosed within the EMI shielding of the outer cabinet, and can be connected to any port within the MPC as desired. This space is also of proper dimension to contain a PDP LSI 11/23 chassis and power supply.

3.1.1.4 Cable Transition Plate. The cable transition plate is at the very bottom of the cabinet accessible from the front (via the door) for routing cables underneath a raised floor. For installation over a nonraised (solid) floor, an optional pedestal type device would have to be built to allow routing cables out of the cabinet. The transition plate consists of several subpanels which allow various types of connectors to be placed on it and optionally added later. The purpose of the transition plate is to allow the standard I/O connectors internal to the cabinet to be interfaced to various types of external interfaces requiring varied styles, sizes, and connector shapes.

3.1.1.5 Power Supply. Several power supplies are present inside the MPC cabinet, depending upon specific configuration. A large switching power supply (300W) provides voltages of +5 at 50A, and ± 12V at 8A, to the card cage (-5V is also supplied via a single component regulator). The power controller is powered by a separate smaller (+5V at 6A mp) supply which operates independently of the larger supply. If a floppy disk or LSI 11/23 chassis is also present, they contain their own power supply. The main AC power entering the cabinet is protected by a 10 Amp, 120 V/AC breaker and is in series with an EMI line filter.

3.1.1.6 Cooling. Cooling within the cabinet is via forced convection, using air drawn from beneath the cabinet base and expelled through the top rear. This arrangement utilizes and enhances the effect of natural rising air convection past the MPC cards.

3.1.1.7 Controls. Controls for the current MPC consist of an AC main breaker controlling power to the fans and an on/off switch for DC power for the cage itself. The present configuration allows the DC power switch on the cabinet containing the master arbitor to control power to all cabinets in a MPC network.

In addition to these controls there are indicator lights to show which cabinet is the master, indicate power on (DC), and whether the master is on for those cabinets slaved to the master.

3.1.2 Exchange Bus (XBUS). The MPC architecture includes a back plane bus to interconnect all ports. The MPC XBUS has a number of unique characteristics which make it superior to the more common forms of bus design found in today's minicomputers. The following paragraphs highlight these unique MPC characteristics.

3.1.2.1 Physical Characteristics. The MPC XBUS is physically short and allows connection of 24 two-board ports in a cabinet. When cabinets are placed side-by-side, an XBUS may be extended across three cabinets to accommodate 72 ports. Further, through facilities of the Inter-Bus

Communications (IBC) subsystem described in paragraph 3.2.1.2 several MPC XBUS configurations may be interconnected to form very large MPC networks.

3.1.2.2 Cycle Time. The MPC XBUS has a fixed cycle time of 500 nanoseconds using tri-state TTL logic for bus drivers. However, since there is no XBUS clock continuously supplying bus cycles, the XBUS actually operates in an asynchronous manner. That is, XBUS cycles only appear on the bus when requested.

3.1.2.3 Bus Width. The MPC XBUS width is 16 bits. Since 8-bit microcomputers are used in th MPC, a double byte load is required to transfer one 16-bit word across the XBUS. This two-byte load occurs in the XBUS interface of the port.

3.1.2.4 Interrupt Resolution. XBUS cycles are granted to requesting ports by a separate XBUS interrupt resolver circuit connected to each port. Interrupt resolution occurs in parallel to data movement across the XBUS. When multiple ports try to acquire XBUS cycles at the same time, the interrupt resolver fields each request on a round-robin basis. When a port has been granted an XBUS cycle, it is able to trasnfer only 16 bits of data before it must request another cycle. This procedure provides for automatic interleaving of data on the XBUS and prevents XBUS domination by any one port.

3.1.2.5 Bus Addressing. The MPC bus utilizes the same circuit paths for both data and address transfers across the bus. Since each 16 bits of data is separately addressed and requires a bus cycle, effective data transfer bandwidth is one-half of minimum XBUS bandwidth or 16 megahertz.

3.1.3 XBUS Control Port. The XBUS Control Port (XBCP) is required on each physical XBUS for the purpose of controlling access to the XBUS. The bus cycle request/grant function of the bus control port determines, or arbitrates, the time slot during which a port has access to the XBUS. Also, the XBCP handles such error conditions as one port attempting to access a nonexisting port or the recovery from issuance of false grants due to noise or other error conditions on the bus. Additionally, the XBCP has the

capability of directly linking up to three XBUSs via a parallel data XBUS cable. If several XBUSs are to be linked directly, then the XBCP arbitrates between them to determine which XBUS has priority. Currently, three XBUSs can be directly linked allowing up to 72 ports to exchange information via directly synchronized bus cycles.

Each XBUS port has a request line to the XBUS Control Port and a grant line from the Control Port. Whenever a port requires control of the XBUS to execute a bus cycle, it issues an asynchronous request to the control port via its own request line. The control port is then responsible for arbitrating among these requests and issuing one and only one grant at a time based upon some type of arbitration scheme. The arbitration scheme for this control port is based upon a priority encoding scheme with several levels of encoding.

Functionally, the XBUS control port consists of arbitration modules, cable transceivers with control logic, and other control logic.

3.1.3.1 XBCP Arbitration Modules. Arbitration modules are the basic building blocks of the Bus Control Port. Each module is an independent circuit linked to other modules for synchronization purposes. Ports on the XBUS link directly to the lowest level of arbitration modules via individual request/grant signal lines. Thus, a 24 port XBUS requires three low level arbitration modules because each module handles eight request/grant pairs; i.e., ports. These low level arbitration modules require one group arbitration module. In addition, if several XBUSs were to be linked directly then a master arbitration module would be required to provide synchronization.

3.1.3.2 XBCP Cable Transceivers and Control. Several XBUSs may be linked via an XBUS cable. This cable transfers bus cycle data between any of three XBUSs under control of the master XBCP. Control logic on each XBCP determines whether its host XBUS is to receive or transmit data and subsequently enables or disables the XBUS cable transceivers. During a bus cycle all three XBUSs become synchronized to the same bus cycle, so that ports may be arbitrarily located in any of the three XBUSs.

3.1.3.3 Control Logic. Control logic exists on the XBCP for the purpose of generating a proper power-up reset and to provide proper error recovery due to exception conditions on the XBUS. Whenever power is first applied to an XBUS, it is necessary for the XBCP to provide a reset signal to all ports. This reset signal assures that both the hardware and software on each XBUS port will be properly initialized. The other function of the control logic is to handle exception conditions on the XBUS. This includes such situations as one port attempting to access a nonexisting port or the recovery from issuance of false grants due to noise or other error conditions on the XBUS. If an attempt is made to access a nonexisting port, the control logic must provide an indication to the requesting port that the bus cycle is to be terminated. If a false grant is issued, the XBCP must decide after an arbitrary time span to terminate the grant signal and process other requests. Currently, this time span is approximately two to three microseconds; i.e., several times the length of a normal bus cycle. Thus, these exception conditions have minimal impact on the throughput of the XBUS.

3.1.4 Bus Interface Logic. The MPC bus is interfaced to each port through a bus interface module resident on each port. This module contains an Input Data Handling Register (IDHR), an Output Data Handling Register (ODHR), and a Bus Address Register (BAR). The IDHR allows data to be written into a port and the ODHR allows data to be written from a port. The BAR contains the address of the destination port for any bus cycle. Additionally, the port's bus interface contains logic to allow the reset/restart of a port when power is applied to the cold system, the state controller associated with the XBUS data transfer protocol, and the circuitry required to request and recognize bus cycle grants. Figure III-02 depicts these components.

3.1.5 XBUS Extension. The current MPC architecture will support a direct connection of three XBUSs with a total of 72 ports. Further extension is not directly supported at the hardware level. The method currently employed to achieve XBUS extension is the interconnection of two independent XBUSs via a high speed communications line. One port on each bus must be dedicated to the support of the communications lines. The firmware resident on those ports is the Inter-Bus Communications (IBC) subsystem. IBC, discussed in

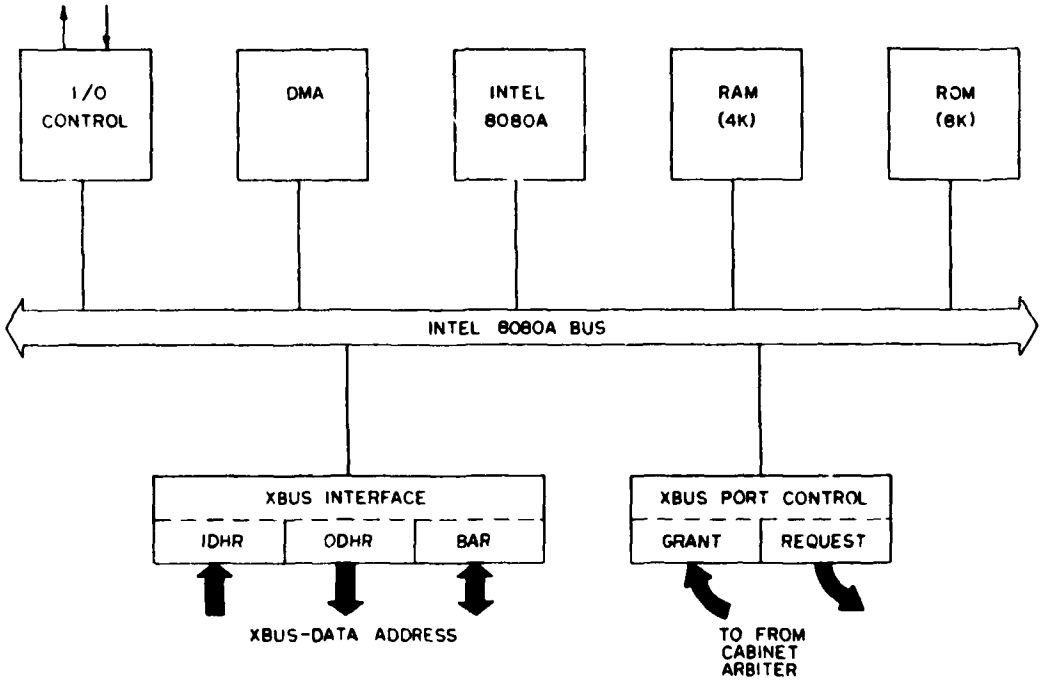


Figure III-02. Typical MPC Port

Section 3.2.1.2, is designed to support cross bus IPC dialogues in a fashion which is transparent to each of the ports engaged in the dialogue.

3.1.6 MPC Option Ports. The MPC uses a two-board approach in fabricating ports to provide added design flexibility. The CPU board contains the Central Processing Unit (CPU), memory, and XBUS interface while the I/O board contains the I/O interface to the attached device. This approach provides the flexibility to develop new I/O interfaces without modifying the CPU board. It also permits changing CPUs or memory without modifying the I/O board.

Both the CPU and I/O boards are fabricated using the Multiwire process that encapsulates all component interconnection circuits in an adhesive substrate instead of the printed circuit approach. This is such a reliable approach that it is used in high density space craft circuits. The cost to produce Multiwire boards is competitive with printed circuit (PC) technology, yields from production runs are higher than multi-level PC boards, and overall production turn-around time is less.

Each board measures 9.5 by 9.5 providing 90.25 square inches or a total of 180.5 square inches of space for electronic components.

3.1.6.1 CPU Board. The CPU board contains the microprocessor, memory, XBUS interface, local port bus, and a connector to the I/O board. Each is described below.

3.1.6.1.1 Microprocessor. The MPC uses the INTEL 8080A microprocessor that executes 500,000 instructions per second. The INTEL 8080A is an 8-bit processor based on NMOS technology and is capable of addressing up to 64K bytes of memory.

3.1.6.1.2 Read Only Memory (ROM). The CPU board contains 8K bytes of eraseable ROM used for program storage. When software is placed in ROM, the resulting firmware takes on the characteristics of hardware and is, therefore, not remotely modifiable. This physical characteristic of ROM

makes its use in the MPC ideally suited to the secure environments typical of Defense Department installations.

3.1.6.1.3 Random Access Memory (RAM). Each CPU board also contains 4K bytes of static RAM for data storage within a port. Both RAM and ROM memory operate on a 500 nanosecond cycle time, matching the cycle time of the INTEL 8080A CPU.

3.1.6.1.4 XBUS Interface. The CPU board also contains logic to interface a port to the XBUS. The XBUS interface contains an Input Data Holding Register (IDHR), an Output Data Holding Register (ODHR), and a Bus Address Register (BAR). The IDHR allows data to be received by a port and the ODHR allows data to be written from a port. The BAR contains the address of the destination port to which data in the ODHR is to be written to.

Additionally, the port's XBUS interface contains logic to allow the reset/restart of a port when power is applied to a cold system, the state controller logic associated with the XBUS data transfer protocol, and the logic required to request and recognize bus cycle grants.

3.1.6.1.5 Port Bus. Each port has its own bus interconnecting its processor, memory, XBUS and I/O interfaces. This means that port operation does not affect the XBUS until sufficient data has been processed and accumulated for transmission to another port. Therefore, the effective bandwidth of the MPC is far in excess of the XBUS. Each port bus also has a bandwidth of 16 megahertz.

3.1.6.2 I/O Board. The I/O board is prewired for several component configurations; however, it is fabricated only with the components required for a specific configuration. The use of component sockets provides this flexibility. The prewired configurations include:

- o Asynchronous/Synchronous Receiver/transmitter
- o Multi-Protocol Communications Controller

o Single/Double Density Floppy Disk Controller

Prewiring of these three types of I/O interfaces also allows fewer spares. That is, instead of maintaining a complete board set for each application, a single CPU board can be used with several I/O boards, as required.

3.1.6.2.1 Asynchronous/Synchronous Receiver/Transmitter. Each I/O board is prewired to use the Western Digital 1931 Asynchronous/Synchronous Receiver/-Transmitter device which interfaces serial data communication channels to a port. The WD1931 is capable of full duplex communications with asynchronous or synchronous systems using character-oriented protocols. Eight selectable clock rates support communication circuits operating at up to 1 million bits per second. This configuration of the Option Module is used to implement the OJ-389 and Inter-Bus Communications (IBC) subsystems. Figure III-03 depicts this particular I/O board.

3.1.6.2.2 Multi-Protocol Communications Controller. Each I/O board is also prewired to accept the Signetics 2652 Multi-Protocol Communications Controller (MPCC). The MPCC is an multichannel device that formats, transmits, and receives synchronous serial data while supporting bit-oriented or byte compatible protocols. Bit-oriented protocols supported include Sychronous Data Line Control (SDLC), High Level Data Control Link (HDLC), and Advanced Data Communications Control Protocol (ADCCP). Character-oriented protocols supported include BI-SYNC and Digital Data Communications Message Protocol (DDCMP). The 1652 MPCC supports line rates of up to 500,000 bits per second. This Option Module configuration is currently used to implement DDCMP interfaces to DEC PDP-11 computers.

3.1.6.2.3 Single/Double Density Floppy Disk Controller. The NEC Microcomputers, Inc., uPD765 Single/Double Density Floppy Disk Controller (FDC) chip contains the circuitry and control functions for interfacing one MPC port to up to four Floppy Disk Drives. The uPD765 FDC is capable of supporting either IBM 3740 single density, or IBM System 34 double density formats, including double-sided recording. Programmable data record lengths

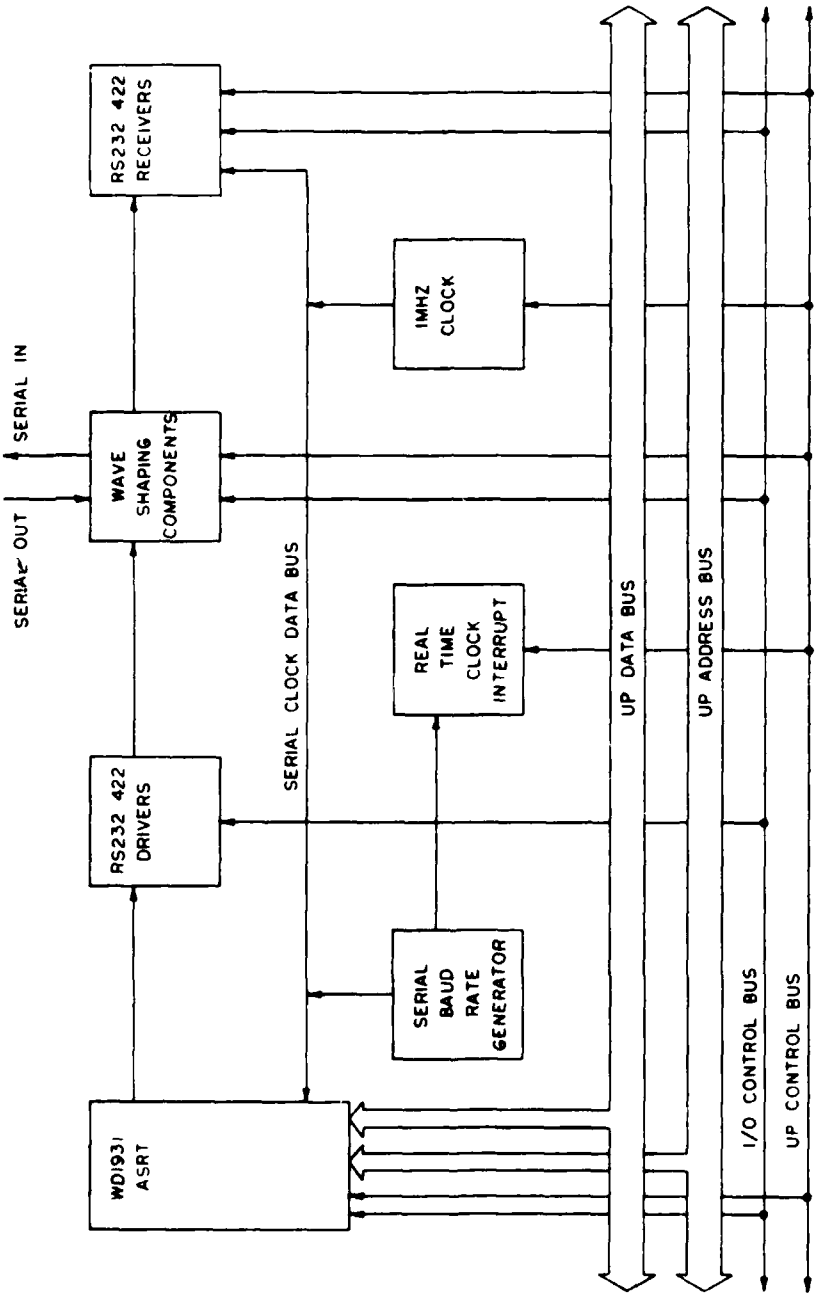


Figure III-03. Asynchronous/Synchronous Receiver/Transmitter

of 128, 256, 512, or 1024 bytes per sector are supported. This Option Module configuration may be used to interface the Floppy Disk which is optional within each MPC cabinet.

3.2 MPC Firmware Description. A combination of system-level and interface specific firmware subsystems reside in the ROM of every MPC Version 1.0 port. All subsystems are contained in ROM with no global memory attached to the bus. In cases where a ports ROM is not sufficient to contain the entire subsystem, the MPC system allows for downloading of modules from other ports into RAM for execution and hence effects a quasi-virtual memory capability. Figure III-04 depicts the MPC firmware subsystems.

3.2.1 System-level Subsystems. System-level control and services for the MPC are provided by its Inter-Port Communications (IPC), Inter-Bus Communications (IBC), Error Detection and Recovery (EDR), and External Control and Monitoring (ECM), subsystems. Debug and diagnostic facilities are provided by the MPC Asynchronous Control Element (MACE). Each subsystem is discussed below. Since IPC is the only subsystem within a port to directly address the bus, the new design the MPC affected major modifications in its structure. For this reason, the current IPC will be discussed first and in greater detail than other subsystems to provide an understanding of its overall function.

3.2.1.1 Inter-Port Communications (IPC). Each MPC port must be able to establish and participate in port-to-port dialogues. The hardware that provides this capability is the XBUS and related circuitry on each port. The software that provides this capability is the IPC subsystem. IPC effectively lies between the bus interface hardware and the remaining MPC subsystems contained within each port. No other subsystem within a port directly addresses the bus interface hardware. In this manner, IPC essentially envelopes the exchange bus and shields the remaining subsystems from the intricacies of the exchange bus interface, thus providing hardware transparency.

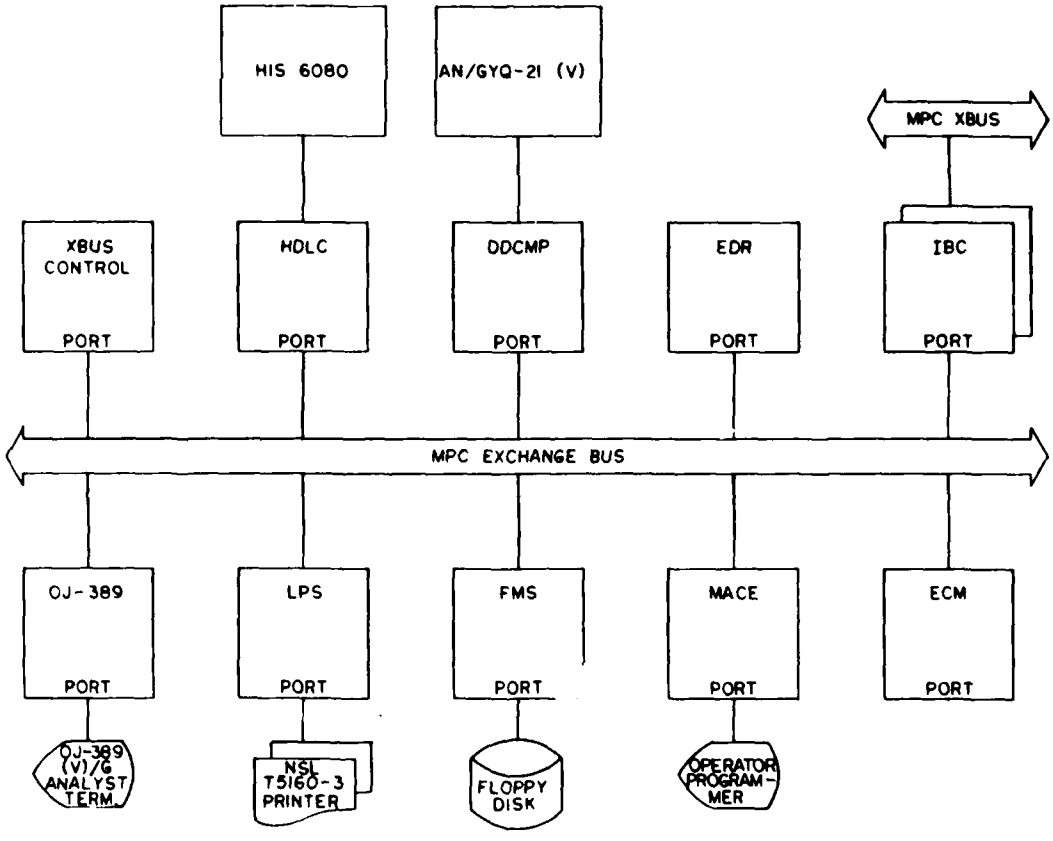


Figure III-04. MPC Firmware Architecture

3.2.1.1.1 Dialogues. A copy of IPC is contained within the ROM of every port in the MPC. When a port subsystem has data or control information to pass to another port it expresses that requirement to IPC which then establishes the dialogue. A dialogue has three components: contact and lockon, data exchange, and termination. Figure III-05 depicts the elements and sequence of each component of an IPC dialogue.

3.2.1.1.1.1 Contact and Lockon. Contact and lockon establishes a port-to-port connection. In this process, an initiating port examines the status of the desired destination port to determine whether it is available, writes a lockon request to that port (contact), and then waits for a lockon response (lockon). If the request is accepted, IPC will retain control of the microprocessors in each connected port. The contact and lockon process ensures that both ports agree to a dialogue and provides synchronization for a subsequent date exchange. Once a dialogue has been established between two ports, all other ports attempting to establish a dialogue will be prevented from doing so until the initial two-port dialogue is completed.

3.2.1.1.1.2 Data Transfer. The physical description of data to be transferred between ports is expressed as a four byte data control word (DCW) passed to IPC from the subsystem requesting the transfer. The actual data transfer is driven by the use of data and control interrupts and is performed under checksum control to ensure communications reliability for data exchange. Furthermore, only two bytes of data may be transferred per bus cycle and a dialogue between ports may not utilize continuous (back to back) bus cycles. Thus, data being transferred between ports is interleaved on the XBUS but the software in any one port is completely unaware of it. Any checksum error will result in data retransmission or termination of the dialogue with appropriate notification to each port. Any number of simultaneous dialogues may be active on the XBUS at any time limited only by the number of ports in the MPC and the capacity of the XBUS.

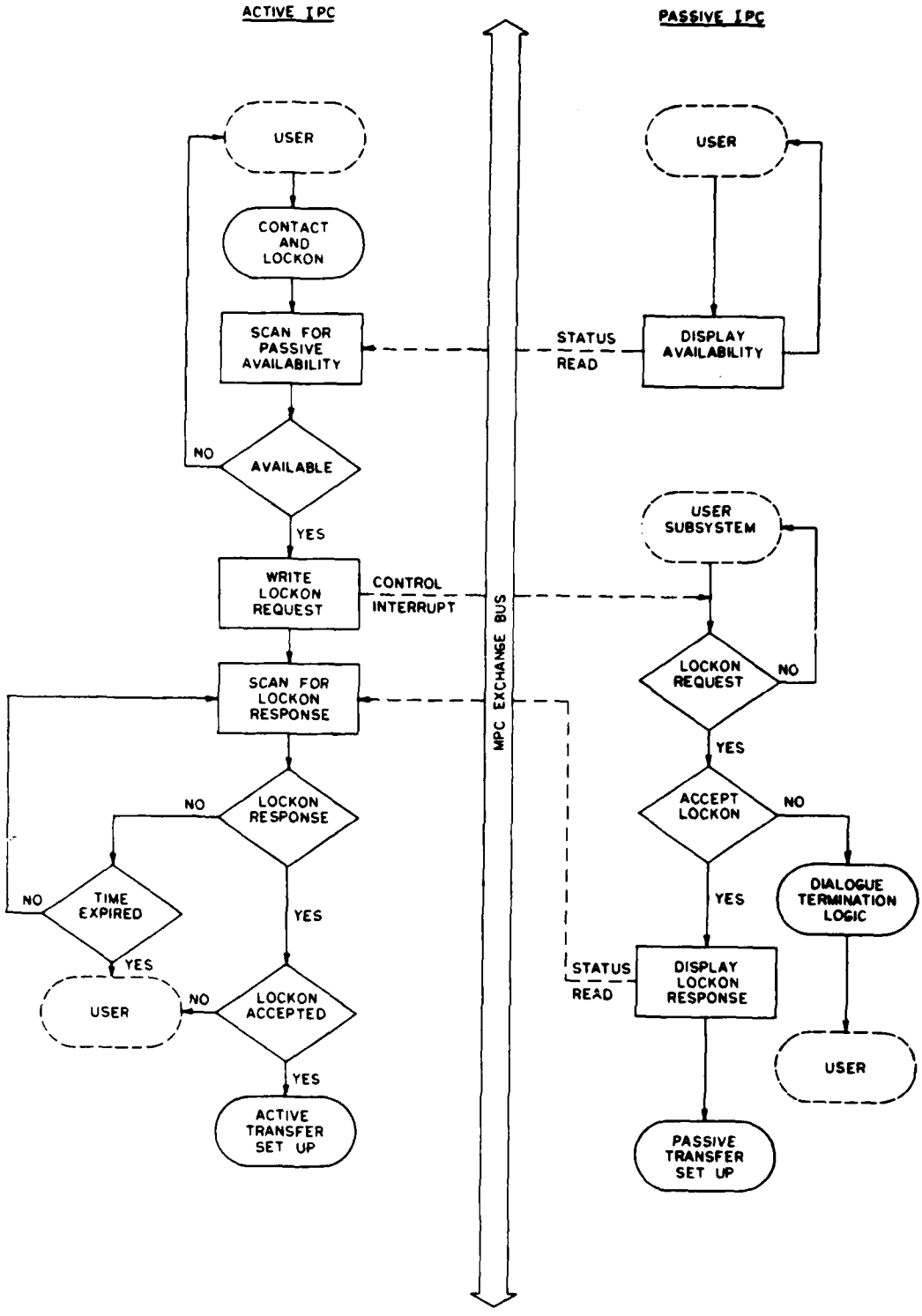


Figure III-05. MPC Dialogue (1 of 3)

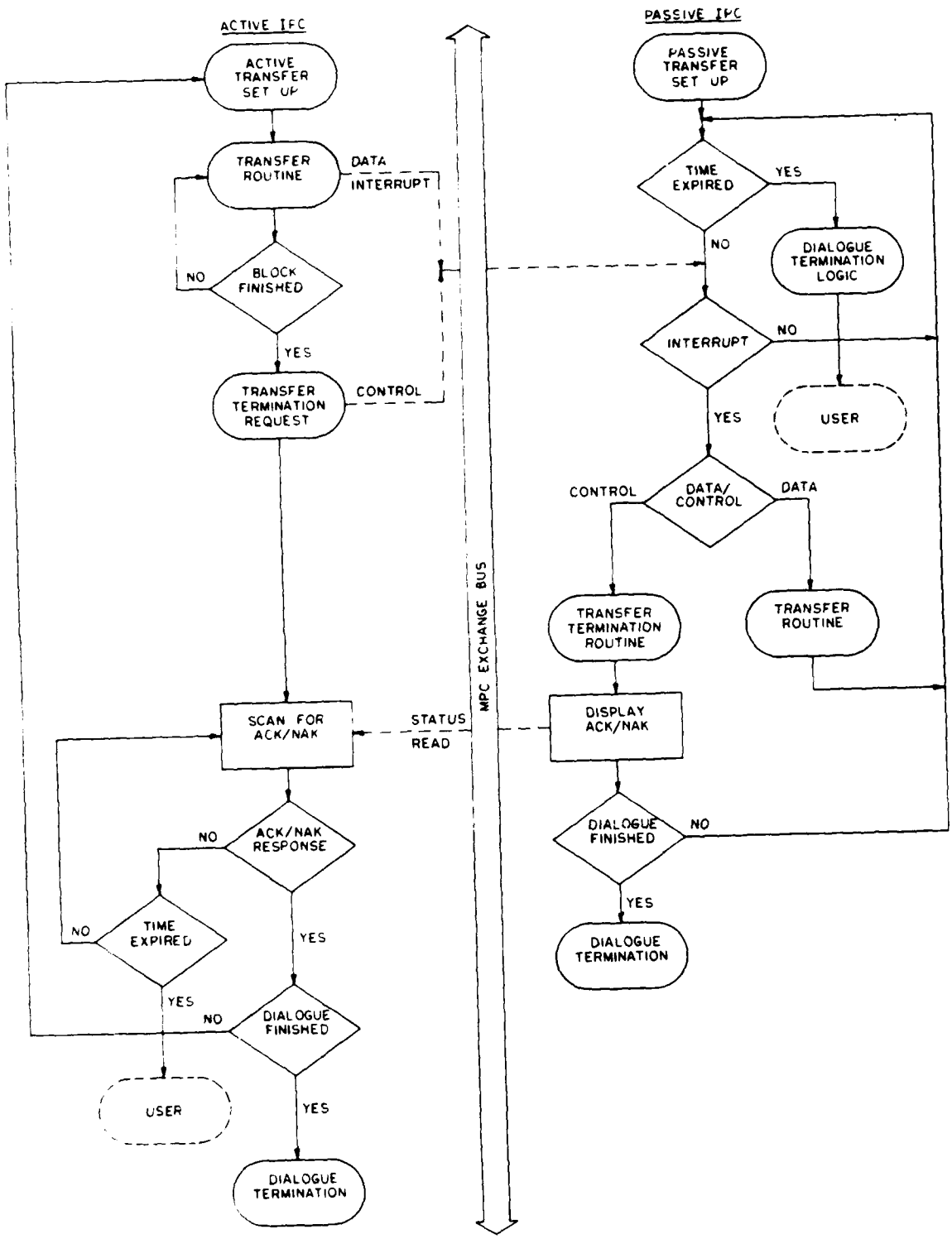


Figure III-05. MPC Dialogue (2 of 3)

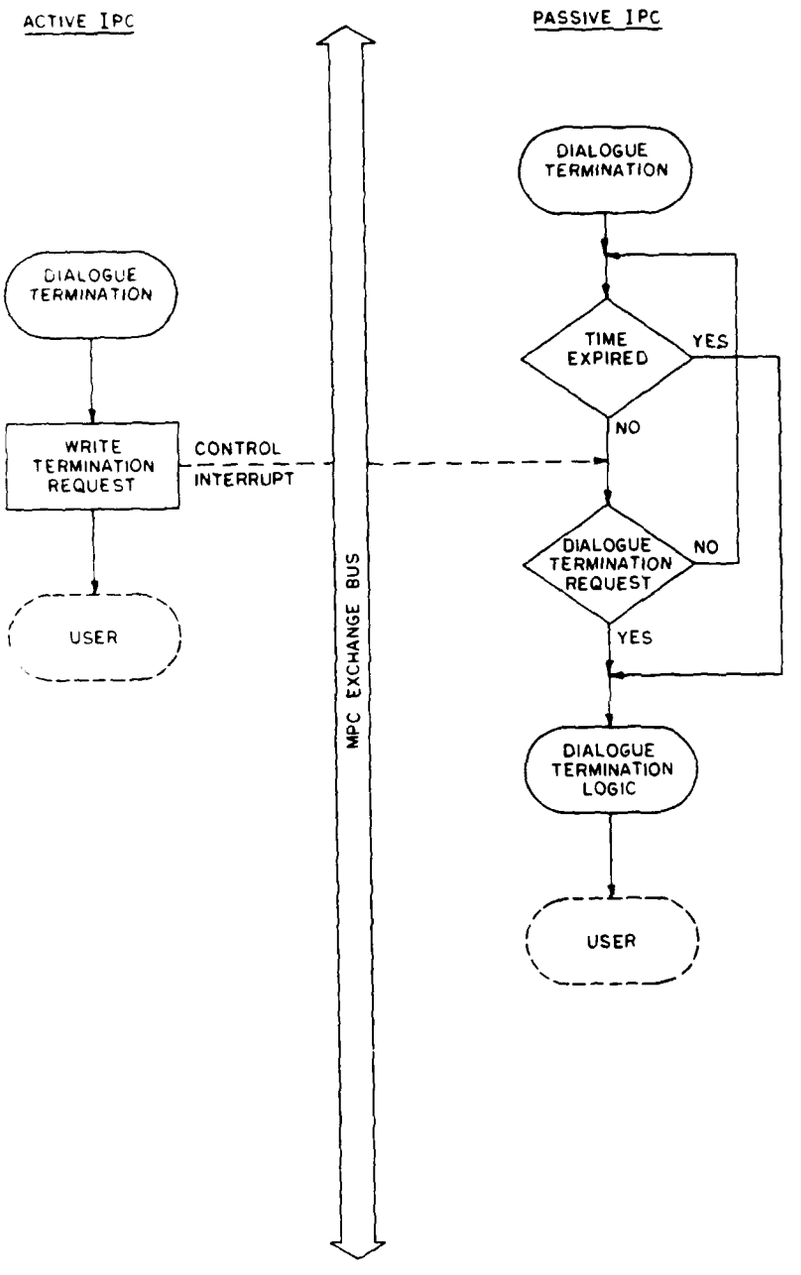


Figure III-05. MPC Dialogue (3 of 3)

3.2.1.1.1.2 Dialogue Termination. The normal termination of a dialogue is signalled by the control byte of the last DCW executed by IPC. At this time, the ports will be disconnected and processing will be restored to the software subsystem executing in each port.

3.2.1.1.2 IPC Components. The IPC subsystem contains two distinct components; the active component and the passive component. The passive component must reside in each port, but the active component is optional, although most ports do require both components. Each component is necessary to a dialogue. The active component of one port controls the dialogue while the passive component of the other port executes in lockstep with the first port as the dialogue proceeds. The active component is invoked as a subroutine call from a subsystem executing within a port. The passive component is executed at the interrupt level in response to interrupts received from the active component of IPC in another port. Behaviorly, the active component operates as a main level extension of a local port subsystem, while the passive component operates as an asynchronous, interrupt level process which executes independently from a local port subsystem. Figure III-06 depicts the concept of active and passive IPC.

3.2.1.1.3 Communication Types. Port-to-port dialogues may be initiated by either an active port or a passive port, depending on which component of IPC will be utilized by the port initiating the dialogue. An active port may initiate a dialogue with a passive port by simply executing a subroutine call to the active IPC component in that port. This type of dialogue is called direct because the initiator port also controls the dialogue. A passive port may also initiate a dialogue, but the dialogue must be controlled by the active port. This type of dialogue is called an indirect dialogue. The passive port cannot initiate the dialogue directly because the passive IPC component in that port can only operate in response to interrupts received from an active port. Thus, the passive port must simply express a desire for the dialogue by conditioning the port status. By prior arrangement, the desired active port must be actively scanning the status of other ports connected to the XBUS in order to detect when a passive port has requested a

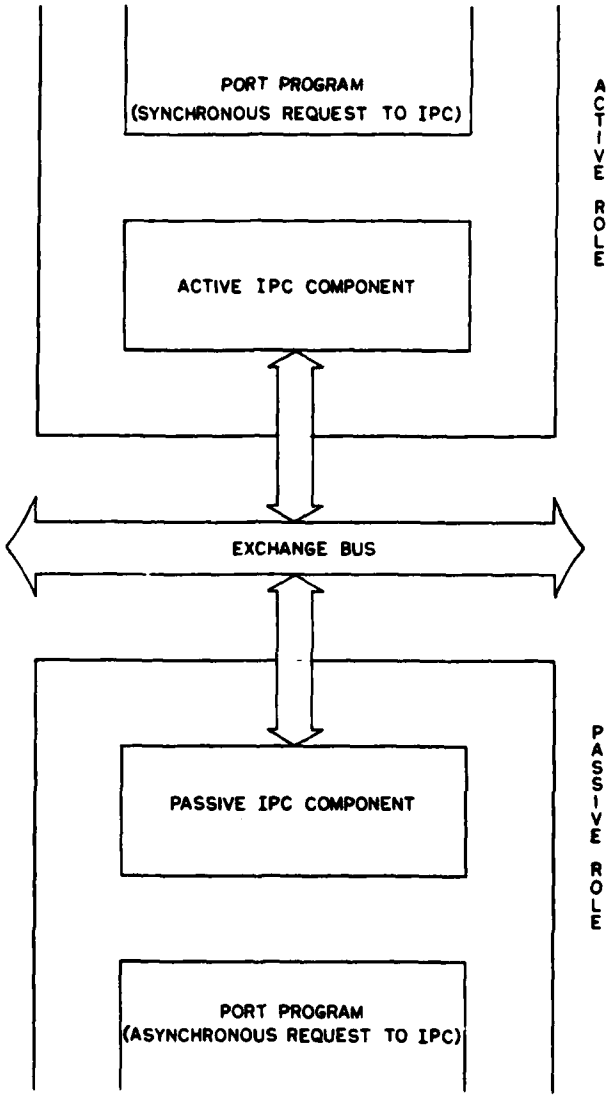


Figure III-06. Port-to-Port Dialogue

dialogue. When the request is recognized by the active port, the active port will then conduct a prearranged dialogue with the passive port.

The direct communications mode is subdivided further into two classes; system direct and network direct. A port may be conditioned to be available for any or all communication types. The system direct mode is generally utilized for general purpose system control messages from subsystems such as the Error Detection and Recovery (EDR) or the External Control and Monitoring (ECM) subsystems. The network direct is utilized by application level subsystems such as the DN-355 or DDCMP subsystems. The subdivsion of the direct mode allows a port to condition itself to receive system control messages but not general network traffic.

3.2.1.1.4 Communications Channels. A Communications Channel is a table which contains all control information needed by IPC in order to conduct a dialogue. Any dialogue between two ports requires a specific Communications Channel in one port to be logically connected to a specific Communications Channel in the other port. A Communications Channel is a long-term control mechanism for communicating between two processes in different ports.

Each port is capable of handling a multi-tasking environment in which several asynchronous processes are executing. Each of these processes may have its own unique communication requirements. Thus, each port may be configured with up to four Communications Channels. Multiple Communications Channels allows IPC an opportunity to multiplex dialogues through the single XBUS interface of each port.

3.2.1.2 IBC Subsystem. The Inter-Bus Communications (IBC) subsystem is designed to provide a XBUS to XBUS link between two MPC systems. Each IBC is identical and the link provided by a pair of IBCs in conjunction with the Inter Port Communications (IPC) software is designed to be transparent to all other MPC subsystems. Transparency is accomplished by mutual arrangement between IPC and IBC subsystems.

3.2.1.3 EDR Subsystem. The Error Detection and Recovery (EDR) subsystem acts primarily as the MPC System Manager. EDR is aware of the software and hardware functions of each port type and provides this information to individual ports as required. EDR maintains various control configuration and status tables in which the dynamic state of each port, console, connected device, or external system is continuously recorded. Logic for the reconfiguration or recovery of various port or port subsets of the MPC is resident in EDR. Commands for the execution of these routines may be received from the MPC operator or from other internal subsystems of the MPC. Provision for the eventual automatic execution of these routines as the result of internal EDR decision logic has been made.

3.2.1.4 ECM Subsystem. The External Control and Monitoring (ECM) subsystem is designed to interface with the MPC monitor console which enables operators and maintenance personnel to monitor MPC operating conditions. ECM utilizes the control configuration and status tables maintained by EDR. ECM commands fall into two categories; those for displaying the current status of MPC components and those for directing the reconfiguration/recovery of various MPC components.

3.2.1.5 MACE Subsystem. The final system-level control subsystem called MPC Asnychronous Control Element (MACE) is an interactive development tool that is integrated with the operation of the ECM subsystem. MACE provides a machine level interface between the MPC programmer or maintenance personnel and any of the various MPC ports. With MACE, any number of MPC ports may be simultaneously synchronized and/or debugged. Any number of MACE ports may be configured. MACE is the subsystem that provides actual hardware control over the MPC monitor console(s) and is able to discriminate between commands to the internal MACE debug facility or to the external ECM subsystem. Commands for ECM are passed through IPC and the MPC XBUS to the ECM port. ECM command logic executed and appropriate display output responses are generated. These responses are transferred back through IPC and the XBUS to the MACE port which performs the actual output on the MPC monitor console.

3.2.1.6 File Management System. The File Management System (FMS) provides standard file management services for any other MPC subsystem. These services include the capability to create and delete files, and the capability to read and write files. The FMS port type is configured as a DEC 11/23 16 bit microprocessor with 84K bytes of memory. The large memory is utilized as cache buffers for file control information. The disk hardware consists of an 80 Mbyte Ampex disk drive which is Tempest Qualified, and a DILOG DQ200 disk controller. Each FMS port type controls a single disk drive. Two FMS ports are configured on the PACER MPC and one on the OISS MPC.

The FMS also includes the File Management Control (FMC) port type which provides operator interface to the FMS and provides a mechanism for locating files for access.

3.2.2 Interface-Specific Application Subsystems. MPC application firmware provides the custom design interface for all communications circuits, processors, or peripherals interfaced with an MPC. Currently, the following interface-specific port application firmware is available or under development.

3.2.2.1 OJ-389(V)/G Subsystem. The OJ-389 port software provides sufficient intelligence to handle line, data, and command manipulation required to control an OJ-389 workstation, in addition to communicating with the host and other MPC subsystems. The major functions of the OJ-389 software are to: (1) receive and process messages from other MPC subsystems or the host and (2) build and send messages for other MPC subsystems or the host and (3) download the OJ-389 control program to the workstation when requested.

3.2.2.2 Local Printer Subystem. The MPC also has a printer subsystem to provide an interface with a Model T5160 local printer which is a TEMPEST approved version of a Centronics 704 printer.

3.2.2.3 HDLC/DN355 Subsystem. The DN355 Subsystem provides the MPC interface to PACER through the HIS-DN355 FEP. The DN355 Subsystem is responsible for maintaining communications between the MPC and the HIS-DN355. This communication is accomplished by the exchange of data frames on 50,000 bits per second (50 KBPS) communications lines between the MPC and the HIS-DN355. Each communications line is terminated in a Receive and Transmit Port pair which manages the communications flow on each line between the MPC and the HIS-DN355. The MPC interfaces to the HIS-DN355 using High Level Data Link Control (HDLC) Remote Network Processor (RNP) procedures.

3.2.2.4 DDCMP Subsystem. The Digital Data Communications Message Protocol (DDCMP) port is designed to interface the MPC with any Digital Equipment Corporation (DEC) computer using the DDCMP protocol. The subsystem provides line start-up and termination, data exchange, and retransmission services over a 56,000 bits per second, full duplex, communication circuit.

3.2.2.5 Special Purpose Communications link (SPCL) Subsystem. The SPCL subsystem for the MPC is currently being designed. It will provide for termination of the SPCL communications link and for CSP/AUTODIN message traffic destined for PACER. The MPC will include a store-and-forward function to hold message traffic and send it on to PACER when that system is available.

SECTION IV
IMPROVED MICROPROCESSOR DESIGN

This section presents a detailed discussion of the Improved Microprocessor Design. The following subsections delineate hardware and software elements of the MPC architecture that have been designed or redesigned to satisfy the specifications established in Section II of this document.

4.1 Design Overview. The Improved Microprocessor Design consists of an integrated group of design changes to the MPC Exchange Bus, Exchange Bus Control Port, Exchange Bus Interface electronics, Inter-Port Communications Subsystem firmware, and the design of a new Inter-Bus Link Port. The following denotes the major design consideration in each component of the Improved Microprocessor Design:

Improved Exchange Bus

- Demultiplexing of data and address lines
- Addition of Function Code Bus

Improved Exchange Bus Interface Electronics

- Decoupling of port processor from the Exchange Bus Interface
- Multi channel DMA

Improved Inter-Port Communications Subsystem Firmware

- Integration and consolidation of the improved hardware

New Inter-Bus Linking Port

- o Asynchronous bus cycle propogation

4.1.1 Improved Exchange Bus. The improved MPC Exchange Bus (XXBUS) will be the common hardware communications media for all ports in an MPC cabinet. Section 4.2 describes this in detail. The most notable feature of the improved XXBUS is the demultiplexing of the data and address lines in the information bus. In addition, a Function Code bus has been incorporated into the information bus to support design changes in the XXBUS interface electronics of each MPC port which involves the formatting of XXBUS information into information packets. The control bus will be simplified to facilitate bus cycle arbitration and error detection. This configuration will increase the bandwidth of the XXBUS by permitting the simultaneous transfer of data, address, and the new Function Code.

4.1.2 Improved Exchange Bus Control Port. The Improved Exchange Bus Control Port (XXBCP) will provide the hardware logic to grant MPC ports access to the XXBUS. Section 4.3 describes this in detail. The improved XXBCP will utilize new arbitration logic that services all bus cycle requests on an equal and independent basis. This improvement, made in harmony with changes to the XXBUS interface electronics, will permit the duration of a bus cycle to decrease to a minimum of 200 nanoseconds and therefore increase the maximum bus cycle rate to 5 megacycles per second. The maximum bandwidth of the XXBUS will then increase dramatically to 80 megabits per second. In addition to increasing the performance capabilities of the XXBCP, the reliability will be increased as well. Through the use of 3 control lines in the XXBUS and a hardware timing device, the XXBCP will be able to detect and recover errors such as a request to access a nonexistant port, or a false grant issued due to noise or other error conditions that may exist on the XXBUS.

The XXBUS and XXBCP will provide the MPC with a powerful high speed hardware communications media for the purpose of transferring data and control information. The remaining aspects of the Improved Microprocessor Design describe improvements to the basic MPC architecture that will optimize the rate at which data can be trasnferred efficiently across this bus.

4.1.3 Improved Exchange Bus Interface Electronics. The improved Exchange Bus Interface Electronics Unit (XXBIU) will provide the interface for an MPC port and its Primary Port Processor Unit (PPPU) to access the improved Exchange Bus (XXBUS). The most significant aspect of the redesign of the XXBIU is the logical decoupling of the PPPU from the XXBUS. The XXBIU will provide all the required logic to gain access to the XXBUS, resolve dialogue contention, transfer data and ensure its integrity in an environment asynchronous to the PPPU. The XXBIU will consist of three hardware modules: Microprogram Control, Direct Memory Access, and XXBUS. These modules will function as integrated yet independent elements under microprogram control to facilitate control and data transfers. To support these 3 modules, the XXBIU will be configured with dedicated Read Only Memory for microinstruction storage, internal buses for routing of data and control, and control lines to support intermodule communication and control. The XXBIU is described in detail in Section 4.4.

This configuration will increase port-to-port bandwidth in the MPC significantly by releasing the PPPU from time consuming overhead processing required to initiate and sustain dialogues. Application subsystems subordinate to the PPPU will therefore have more processing power at their disposal. The result of these improvements, made in harmony with changes to the Inter-Port Communications Subsystem, will permit highly efficient port-to-bus data transfers at a 10 megabit per second rate.

4.1.4 Improved Inter-Port Communications Subsystem Firmware. The improved Inter-Port Communications (IPC) subsystem will provide the software interface to consolidate the improved hardware components of the Improved Microprocessor Design. The most significant aspect of the redesign of the IPC subsystem is the distinct separation of IPC into two logical levels of operation.

The first level of IPC, IPC Level One (IPC L1), will consist of a group of microprograms residing in the XXBIU. This micro software will perform all of the overhead processing associated with dialogue initiation and data transfer asynchronously to the PPPU. The second level of IPC, IPC Level Two (IPC L2), will consist of firmware residing in the Read Only Memory (ROM) of the port. This level of IPC will execute under the domain of the PPPU and initiate IPC L1 microprograms in a structured manner to perform port-to-port dialogues.

The improved IPC subsystem firmware, utilizing the hardware resources of the XXBIU, will provide a highly efficient communications interface. This configuration will allow port resident application subsystems to devote more processing power to their individual requirements and therefore permit the MPC architecture to address integrated problems of a greater magnitude. The IPC is further described in Section 4.5.

4.1.5 Inter-Bus Linking Port. The new Inter-Bus Linking Port (XXBLP) will provide an asynchronous hardware connection to link two or more MPC XXBUSs together. The XXBLP will be an instrumental component of the improved MPC architecture which will support application environments where more than 24 MPC ports are required to solve a problem. The most notable feature of the new XXBLP will be the decoupling of XXBUS's at the bus cycle level. The XXBLP will transmit and receive data and control information contained in an individual bus cycle to physically isolated, but logically connected XXBUSs.

The XXBLP will consist of an XXBUS transmitter, receiver, Inter Bus Link (IBL) cable, and IBL cable electronics. A queue to buffer Information Packets (IP) will provide the asynchronous decoupling of XXBUSs and produce a pipeline effect for transfers across the IBL cable. The XXBUS transmitter and receiver will be configured to allow variable routing paths between XXBUSs to prevent possible bottlenecks from degrading the overall performance of a MPC network. In addition, because the XXBUS transmitter and receiver logic will propogate each IP asynchronously at the bus cycle level, the XXBLP will be able to multiplex an infinite number of dialogues concurrently without regard for individual dialogue synchronization.

This configuration will allow system planners to design and tailor an integrated network of MPC XXBUSs to solve unique, large scale problems. The use of XXBLPs would permit network growth with a potential for up to 4096 ports (64 XXBUSs) in an individual network, where each individual bus and its associated ports would operate at full bandwidth oblivious to the remaining components of the network. The XXBLP is further described in Section 4.6 of this document.

4.2 The XXBUS Information Packet. Information that is exchanged between the ports of an MPC via XXBUSes and XXBUS linking ports, (described later in this document), is transferred in Information Packets (IP). Each IP corresponds physically to the amount of information that is written onto an XXBUS during a single bus cycle and logically to the smallest amount of information that can support communication between any two ports in a XXBUS network. Figure IV-01 shows the structure of the XXBUS Information Packet. As shown, the IP consists of the data, address, and function code fields. The following is a description of each of these fields and how they support communications between the ports of an MPC.

4.2.1 XXBUS Information Packet Data Field. The IP data field is 16 bits in width which corresponds to the width of a port system data bus. The IP data field contains the data that is to be transferred between processing elements of MPC ports.

4.2.2 XXBUS Information Packet Address Field. The XXBUS IP address field consists of 14 bits and is partitioned (Figure IV-01) into three fields: the PATH I.D., BUS I.D., and the PORT I.D. Table IV-01 provides a detailed description of each of these fields.

4.2.3 XXBUS Information Packet Function Code Field. The XXBUS IP function code field consists of 7 bits. The function code is used to specify the function of the addressed port that is to operate on the IP data field.

4.2.4 Improved MPC Exchange Bus. The Improved MPC Exchange Bus (XXBUS) includes all of the bused and radially connected information and control circuit paths that provide for the transfer of information between MPC ports of a XXBUS card cage. Each XXBUS card cage contains 25 port slots, one for the XXBUS control port and 24 for any type or mix of MPC ports. Figure IV-02 shows the XXBUS to contain four sets of bused parallel circuit paths connecting these port slots. Three of these sets, the data, address, and function code buses, are referred to collectively as the information bus. The fourth is referred to as the XXBUS control bus. Along with these bused paths each

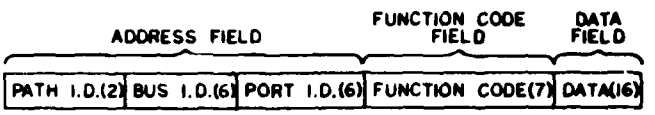


Figure IV-01. XXBUS Information Packet

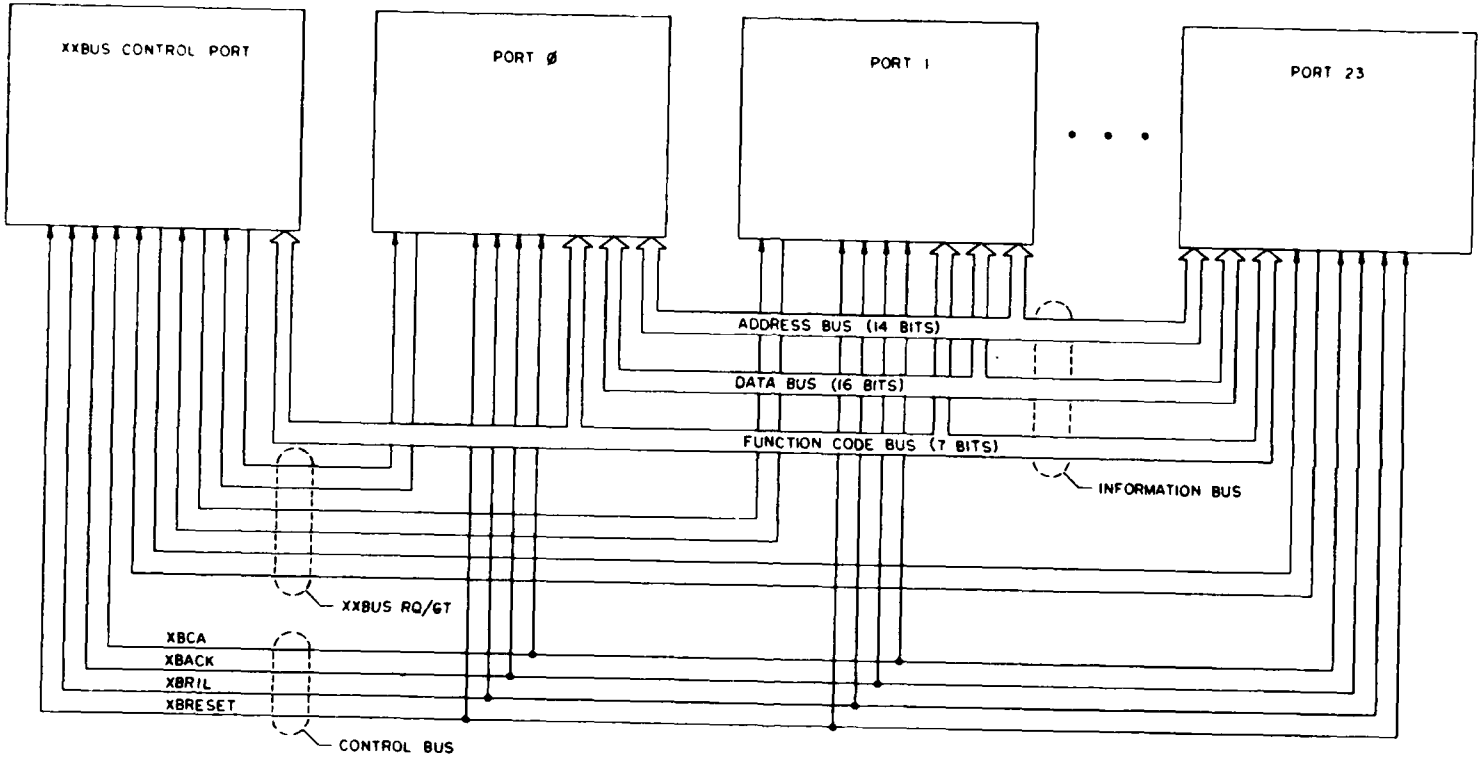


Figure IV-02. XXBUS Functional Diagram

of the 24 port slots has a pair of dedicated control lines, XXBUSRQ/XXBUSGT, which are connected to the XXBUS Control Ports Arbitration Unit.

4.2.4.1 XXBUS Information Bus. The XXBUS information bus is designed to carry a complete information packet during each bus cycle. Although each of the circuit paths comprising the information bus are identical in timing and electrical characteristics, for discussion purposes the XXBUS information bus is partitioned into three additional buses: data, address, and function code buses. These correspond to the data, address, and function code fields of the information packets.

4.2.4.1.1 Data Bus. The Data Bus will consist of 16 parallel paths which connect the 24 port slots. These 16 parallel paths correspond to the 16 bits of the data field of the XXBUS information packet.

4.2.4.1.2 Address Bus. The Address Bus will consist of 14 parallel paths which connect the 24 port slots. These 14 parallel paths correspond to the 14 bits of the address field of the XXBUS information packet.

4.2.4.1.3 Function Code Bus. The Function Code Bus will consist of the 7 parallel paths which connect the 24 port slots. These 7 parallel paths correspond to the 7 bits of the function code field of the XXBUS information packet.

4.2.4.2 XXBUS Control Bus. The XXBUS Control Bus will consist of the control lines used to synchronize XXBUS port transmitters and receivers during XXBUS cycles. They are also connected to the XXBCP which uses them to coordinate the operations of the arbitration unit with XXBUS cycle operations. The XXBUS control bus consists of four control lines: XBCA, XBACK, XBIRL, and XBRESET.

4.2.4.3 XXBUSRQ and XXBUSGT Control Lines. Each port slot has a pair of XXBUSRQ/XXBUSGT lines connecting it to the XXBCP Arbitration Unit. The XXBUSRQ lines are used by the ports of the XXBUS to request the use of the XXBUS. The XXBUSGT lines are used by the XXBCP to grant the use of the XXBUS to a requesting port.

4.3 THE XXBUS Control Port. Since only a single port may use the XXBUS at any one time, a mechanism for establishing when a port may use the XXBUS is required. This section describes the XXBUS Control Port (XXBCP) which provides this mechanism along with those used for detecting and recovering faulty bus cycles and for aiding system initialization during power up and system reset.

4.3.1 General Description. Each port on the XXBUS will have a dedicated pair of signal lines, XXBUSRQ/XXBUSGT. The XXBUSRQ line will be used to make a request to the arbiter and the XXBUSGT line will be used by the arbiter to grant use of the XXBUS to the port. The XXBUSRQ lines are sequentially scanned and latched by the arbiter when they become active. The scanning operation will provide a Last-Look-At-Lowest-Priority (LLLP) method of determining which requesting port will be granted use of the XXBUS.

Principle characteristics of the XXBCP design are:

- The arbiter will be modular consisting of nine modules organized into three levels of arbitration. These nine modules can provide arbitration of 24 ports however, more modules can be added to expand this number up to 40. This provides for a total of 64, which can be added without adding another level of arbitration.

- Worst case request to grant delay time when the XXBUS is not busy will be less than 200 nanoseconds. Worst case queued grant-to-grant delay will be less than 50 nanoseconds.

- The XXBUS Control Port will detect and provide recovery from faulty bus cycles caused by requesting transmitter failing to respond to grant or by the receiver failing to engage transmitter.

- The XXBCP will provide the proper XXBUS reset signal during power up. This reset can also be triggered manually or by writing a special command Function Code to the XXBUS.

4.3.2 XXBUS Control Port Structure. Figure IV-03 shows the functional block diagram of the XXBCP. As shown the XXBCP will consist of the arbitration unit, the XXBUS cycle sequencer, the XXBUS time out timer, and the XXBUS reset unit. Associated with these major functional units are the following control lines: XXBUSRQO-XXBUSRQ23, XXBUSGTO-XXBUSGT23, GRANTED, QUEGT, OUTGT, STARTTIMER, TIMEOUT, and XXBCP reset. This section describes these units and how they are organized to support the functions of the XXBCP.

4.3.2.1 XXBCP Arbitration Unit. The arbitration unit will consist of nine logically identical modules organized hierarchally into three levels of arbitration (see Figure IV-04). The first level will consist of six modules. Each of the Level 1, or Port Arbiters can arbitrate over four ports. These six Port Arbiters are divided into two groups of three modules each. The Port Arbiters in each group must make a request to, and receive a grant from, the Level two or group module before issuing a grant. Likewise, the two group arbiters prior to issuing a grant to a port arbiter must make a request to and receive a grant from the Level three or master module.

Each module will consist of two separate elements. The first will perform the function of scanning the request lines and queuing the grant to be issued. The second will perform forwarding requests to the next level of arbitration. This arrangement will permit the parallel operation of request forwarding, scanding, and latching the request.

The operations of these modules and the XXBUS will be coordinated by the three control lines, GRANTED, QUEGT, and OUTGT. The GRANTED control line will be wired to all of the Port Arbiters and will be used to indicate when the arbiter is issuing a grant. The OUTGT control line will also connect to all of the Port Arbiters and will be used by the XXBUS cycle sequencer to synchronize the outputting of grants with the completion of XXBUS cycles. The QUEGT control line will be connected to all of the modules of the arbiter and be used by the XXBUS cycle sequencer to release the arbiter from its granting state.

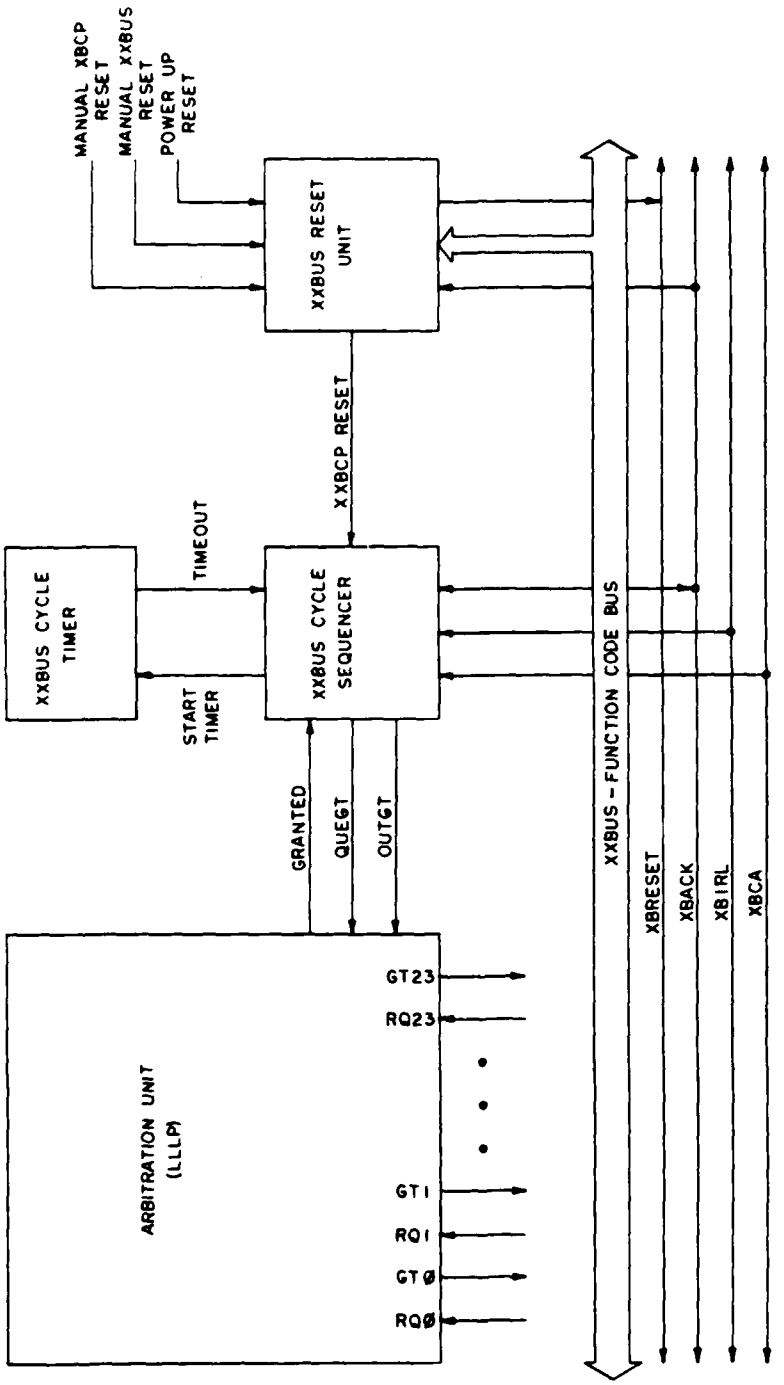


Figure IV-03. XXBUS Control Port Functional Block Diagram

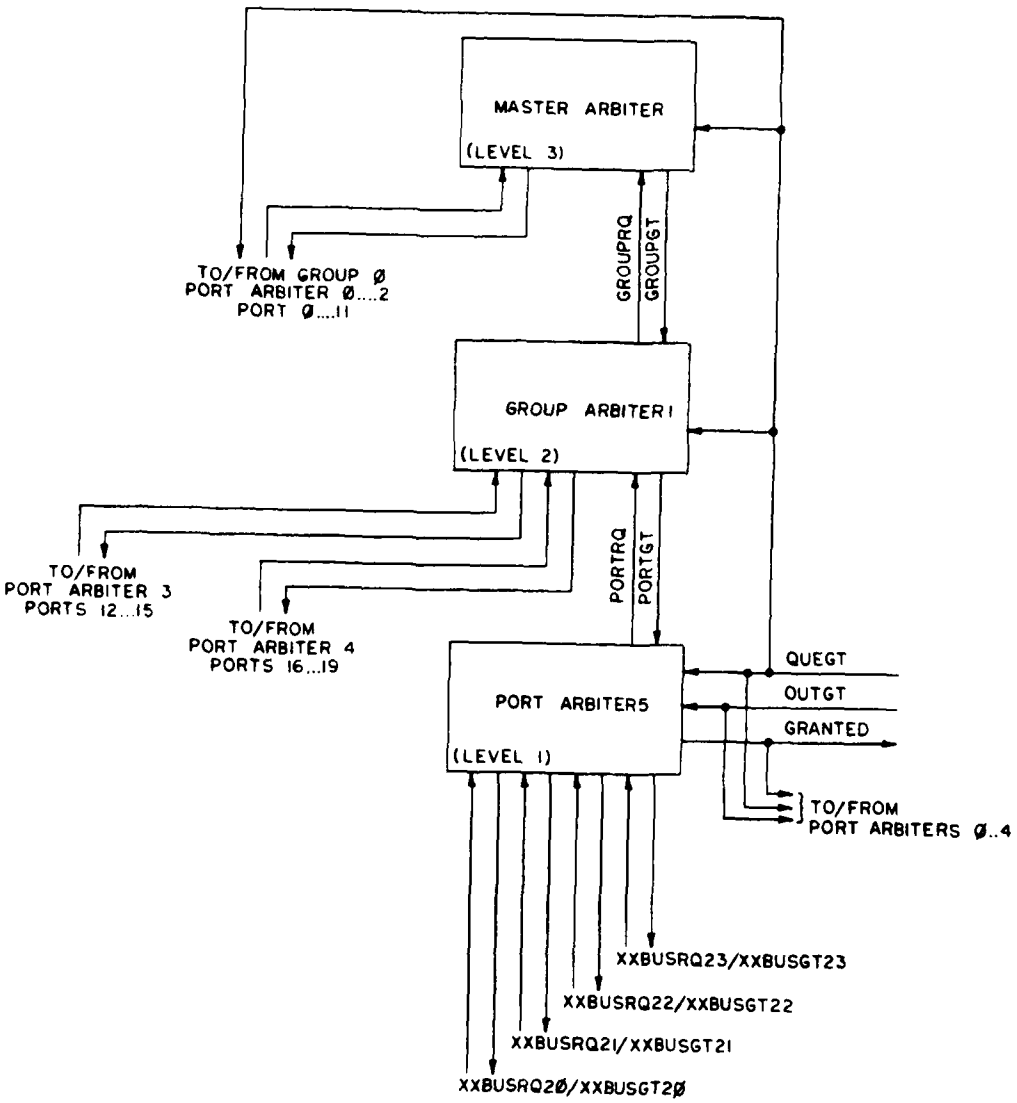


Figure IV-04. XXBCP Arbitration Unit

4.3.2.2 XXBUS Cycle Sequencer. The XXBUS cycle sequencer will control the sequencing of XXBUS cycle initiation and completion with the request scan and grant queuing operations of the arbitration unit. The logic of this control will be implemented as an Algorithmic State Machine (ASM) which coordinates its activities with the arbiter using the control lines GRANTED, QUEGT, and OUTGT; and the XXBUS using the control lines XBCA, XBACK, and XBIRL.

Associated with the XXBUS cycle sequencer is the XXBUS cycle timer which will be used to time out bus cycles that fail to take place or to complete. The control line start timer will be used to enable and reset the timer and the control line time out will be used by the timer to indicate the time out condition.

4.3.2.3 XXBUS Reset Unit. The XXBUS reset unit will provide the proper reset signals to the XXBUS and/or the XXBCP. These reset signals can be triggered either by power up, manually, or by writing the appropriate command Function Code to the XXBUS. The control line XBRESET will be sourced by the XXBUS reset unit and will be used to reset all the ports on an XXBUS. The XBACK line will time the decoding of the execution codes. The manual XXBCP, and XXBUS reset lines will be connected to hardware switches.

4.3.3 XXBUS Control Port Operations. The operations of the XXBUS control port will include arbitrating requests for use of the XXBUS, sequencing the operation of the arbiter with the operation of the XXBUS, detecting and correcting faulty XXBUS cycle operations, and providing the proper reset signals during power up and system reset. The following is a description of these operations presented with respect to the operations of the arbitration, sequence control, and reset control units.

4.3.3.1 XXBCP Arbitration Unit Operations. The XXBCP arbitration unit will perform the operation of arbitrating requests for use of the XXBUS by the various processing and linking ports of the XXBUS. The arbitration operation includes scanning for XXBUS requests, latching and forwarding incoming requests, queuing latched requests, and outputting the XXBUSGT.

The arbiter will receive requests and issue grants via the 24 pairs of XXBUSRQ and XXBUSGT lines. The arbiter will coordinate with the XXBUS cycle sequencer via the control lines GRANTED, OUTGT, and QUEGT. A description of each of these signals is provided in Table IV-01.

Figure IV-05 is an example of the relative timing of specific arbiter signal lines and serves to illustrate the operation of the XXBCP arbitration unit. The following is a description of this operation made in reference to this specific example:

The sequence of events begins with all port, group, and mater modules scanning. That is, none of the XBUSRQ lines are active and the XXBUS is idle. During this time the control lines GRANTED and QUEGT will be inactive and control line OUTGT will become active.

- When an XXBUSRQ line becomes active, this request will be immediately forwarded to the master arbiter. In the diagram, XXBUSRQ 23 becoming active causes PORTRQ 5 to become active which in turn causes GROUPRQ 1 to become active (see (1)).

- When the master arbiter scans a GROUPRQ line and sees it active it will immediately stop scanning and make the corresponding GROUPGT line active. In the diagram GROUPGT 1 will become active (2).

- While the request is being forwarded, the group arbiter's scanner will be sequencially looking at the PORTRQ lines connected to it. When the scanner sees the active PORTRQ line it will also stop scanning. The group arbiter will not, however, make the corresponding PORTGT active until its GROUPGT line becomes active. In Figure IV-05, GROUP Arbiter 1 does not make PORTGT 5 active until it has scanned and latched PORTRQ 5 and GROUPGT 1 becomes active (3).

FIELD	SOURCE	DESTINATION	DESCRIPTION
PATH I.D.	PPPU	XXBLP	Used by PPPU to specify PATH IP is to take. Used by XXBLP along with BUS I.D. to determine if IP should be accepted and forwarded.
BUS I.D.	PPPU	XXBLP	The BUS I.D. specifies the bus location of the destination port. If the BUS I.D. falls within a range of BUS I.D. the XXBLP will, if the PATH I.D. matches, accept the IP.
PORT I.D.	PPPU	XXBIU	Used by PORT in conjunction with BUS I.D. to determine if IP should be accepted.

TABLE IV-01. XXBUS IP Address Fields Definitions

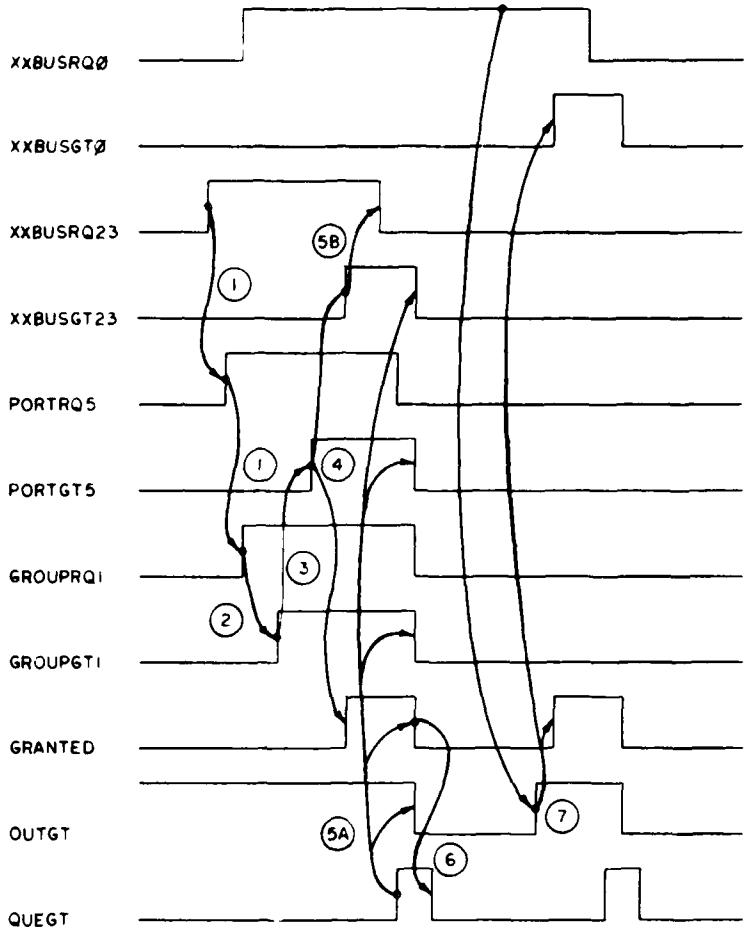


Figure IV-05. XXBUS Arbiter Scan-Request-Grant Timing

- Again, while the request is being forwarded, the Port Arbiter's scanner will be sequencially looking at the XXBUSRQ lines connected to it. The Port Arbiter will stop scanning when it sees XXBUSRQ active. If and when OUTGT and its PORTGT line become active, the Port Arbiter will then proceed to make the corresponding XXBUSGT line active. When XXBUSGT becomes active, the granted line which is wired to all the Port Arbiters will be made active by the granting Port Arbiter. In the example, OUTGT is already active therefore Port Arbiter 5 makes XXBUSGT 23 active upon seeing PORTGT 5 becoming active (4).

- The master and involved Group and Port Arbiters will remain in the granting state until QUEGT becomes active.

- Upon receiving an XXBUS grant from the Arbiter, the granted transmitter will proceed, if operating properly, with a bus cycle. The start of the XXBUS cycle will be sensed by the XXBUS cycle sequencer which will then proceed to make QUEGT active. When QUEGT becomes active, the granting arbiter modules will leave their granting states and return to their scanning states. In the example when QUEGT becomes active, Port Arbiter 5, GROUP Arbiter 1, and the master arbiter will deactivate their grant lines and resume scanning (5A).

  Also, OUTGT will become inactive at this time and will remain inactive during the duration of the bus cycle.

- Upon receiving an XXBUS grant from the Arbiter, the transmitter if operating properly will proceed with a bus cycle. When it does so, its corresponding XXBUSRQ line will become inactive (5B).

- When the granting Arbiters leave their granting state, the granted line will become inactive. This action will indicate to the sequencer that the arbiter has resumed scanning and therefore will make QUEGT inactive again.

It should be noted that while the master and involved Group and Port Arbiters are in their granting state, the other Group and Port Arbiters will continue to scan their request lines. If a XXBUSRQ to a non-involved Arbiter is received, that request will be immediately forwarded to the master module where it will wait until the master module resumes scanning. When the Master Arbiter resumes scanning the pending GROUPRQ will be sensed and the corresponding GROUPGT and PORTGT lines will become active. The Port Arbiter will hold off making the corresponding XXBUGT line active until OUTGT becomes active again at the end of the previous bus cycle (6).

- When OUTGT becomes active again, the Port Arbiter with the queued grant will make the corresponding XXBUSGT line active. In the figure, XXBUSRQ 0 became active shortly after XXBUSRQ 23 did. The request generated by XXBUSRQ 0 is forwarded to the master module via PORTRQ 0 and GROUPQ 0. When the Master Arbiter resumes scanning, it sees the active GROUPRQ 0 line; the Master Arbiter stops scanning and makes GROUPGT 0 active which in turn causes PORTGT 0 to become active. With PORTGT 0 active, OUTGT becoming active is the only remaining condition for XXBUSGT 0 becoming active (7).

4.3.3.2 XXBUS Cycle Sequence Operations. The XXBUS cycle sequence will perform the operation of sequencing the arbiter with the XXBUS. Figure IV-06 shows the relative timing of the control signals associated with the operation of the XXBUS cycle sequencer. The following is a description of the operation of the XXBUS cycle sequencer given in relationship to the relative timing of these signals:

- The sequence begins with the XXBUS idle. When the XXBUS is idle the sequencer will make OUTGT active. When it receives an XXBUS request it will respond immediately by issuing the corresponding grant and making granted active (see (1) in diagram).

- When granted becomes active, the sequencer will proceed to start the XXBUS cycle time out by activating the control line STARTTIMER (2).

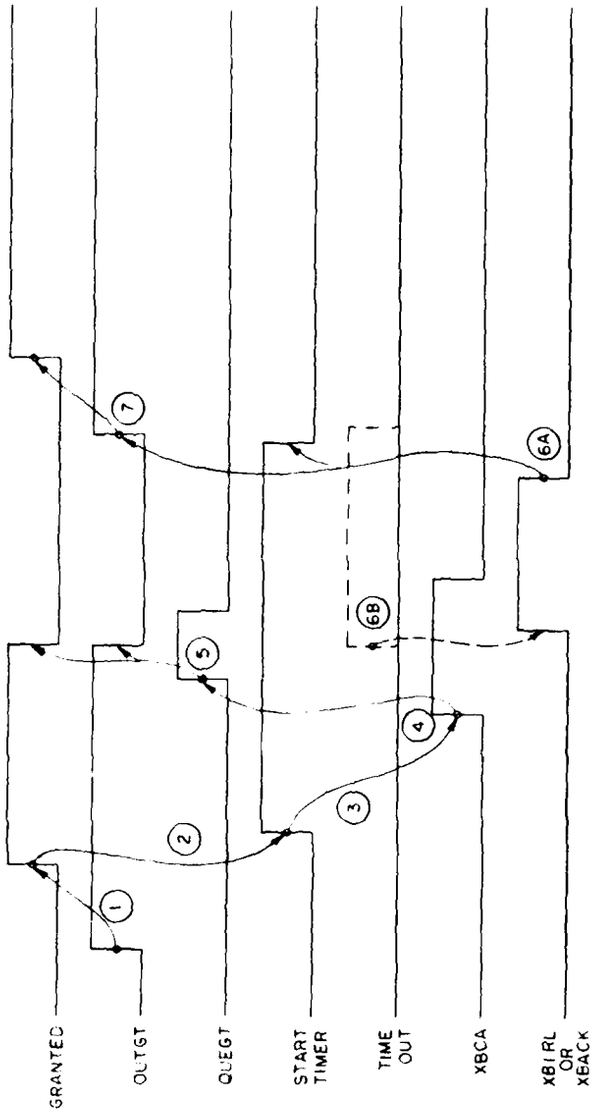


Figure IV-06. XXBCP XXBUS Sequence Timing

- The sequencer will remain in the STARTTIMER state until the transmitter receiving the grant starts the XXBUS cycle by making XBCA active (3).

- When XBCA becomes active, the sequencer will know that the transmitter received the grant. The sequencer will then proceed to make QUEGT active which will cause the Arbiter to resume scanning and to queue the next grant (4).

- When the arbiter resumes scanning, the sequencer will deactivate OUTGT and keep it inactive until the completion of the XXBUS cycle; the granted line will become inactive (5).

- At this point, the sequencer will wait for the normal completion of the bus cycle or for the time out condition to occur. A normal bus cycle will proceed with the transmitter engaging a receiver via the control lines XBCA/XBIRL/XBACK as shown in Figure IV-06. When all of these signals become inactive the bus cycle will be completed and the sequencer will proceed to reset the time out timer and reactivate OUTGT (6A).

- If the control line TIME OUT becomes active, then the sequencer will engage the transmitter by providing the proper sequencing of XBACK. This action will enable the transmitter to complete the bus cycle (6B). In both cases upon completion of the bus cycle, OUTGT will be returned to its active state and start timer will be returned to its inactive state.

- At this point, the arbiter will issue the next grant if one is queued (7).

4.4 Exchange Bus Interface Unit (XXBIU). The XXBIU is a microprogrammable interface control unit that will provide the interface between the Primary Port Processing Unit (PPPU) and the new MPC Exchange Bus (XXBUS) (Figure IV-07). The general architecture of the new XXBIU design is illustrated in Figure IV-08. The improved XXBIU will consist of microprogram memory, a Microprogram Control Module, DMA Module, and a XXBUS module. Independently controlled data paths and control lines will be utilized to interconnect the XXBIU modules internally, as well as to the Primary Port Processing Unit, and the new MPC Exchange Bus. These architectural features contribute to the realization of the design requirements and are further discussed in the following paragraphs.

The Microprogram Memory (MM) of the XXBIU will be dedicated to the storage of Microinstruction directives (MI directives). Having a dedicated MM will eliminate the latency periods introduced by multiplexing which is required if operands and MI directives share the same memory. This also means that the more complicated logic needed to implement the multiplexing scheme will not be required. Through the addition of a mechanism to support the pipelining of MI directives, operations related to MI directive transfers and operand transfers can take place in parallel.

The microprograms will remain static and cannot be dynamically altered by the XXBIU. This simplifies the addressing hardware and will also ensure integrity of the microprograms by preventing an inadvertent alteration due to software or hardware addressing errors.

The hardware addressing control imposes partitioning of the MM into three fixed groupings. These groupings will be organized hierarchically into three levels. The first and highest level divides the memory into 32 equal sections, or microprograms. Each of the 32 microprograms are divided into 8 equal subsections, or microprogram modules. Each microprogram module contains 8 microinstruction words, representing the lowest level of MM addressing. Each microinstruction word may contain one or two MI directives with

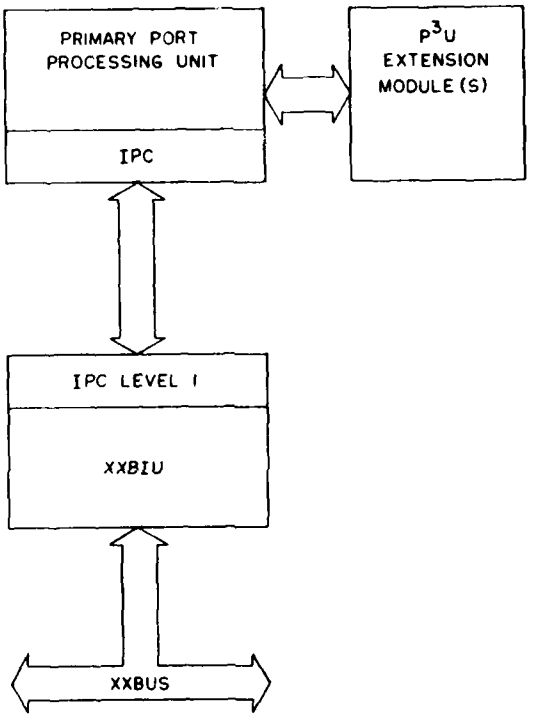


Figure IV-07. XXBIU Interface Unit

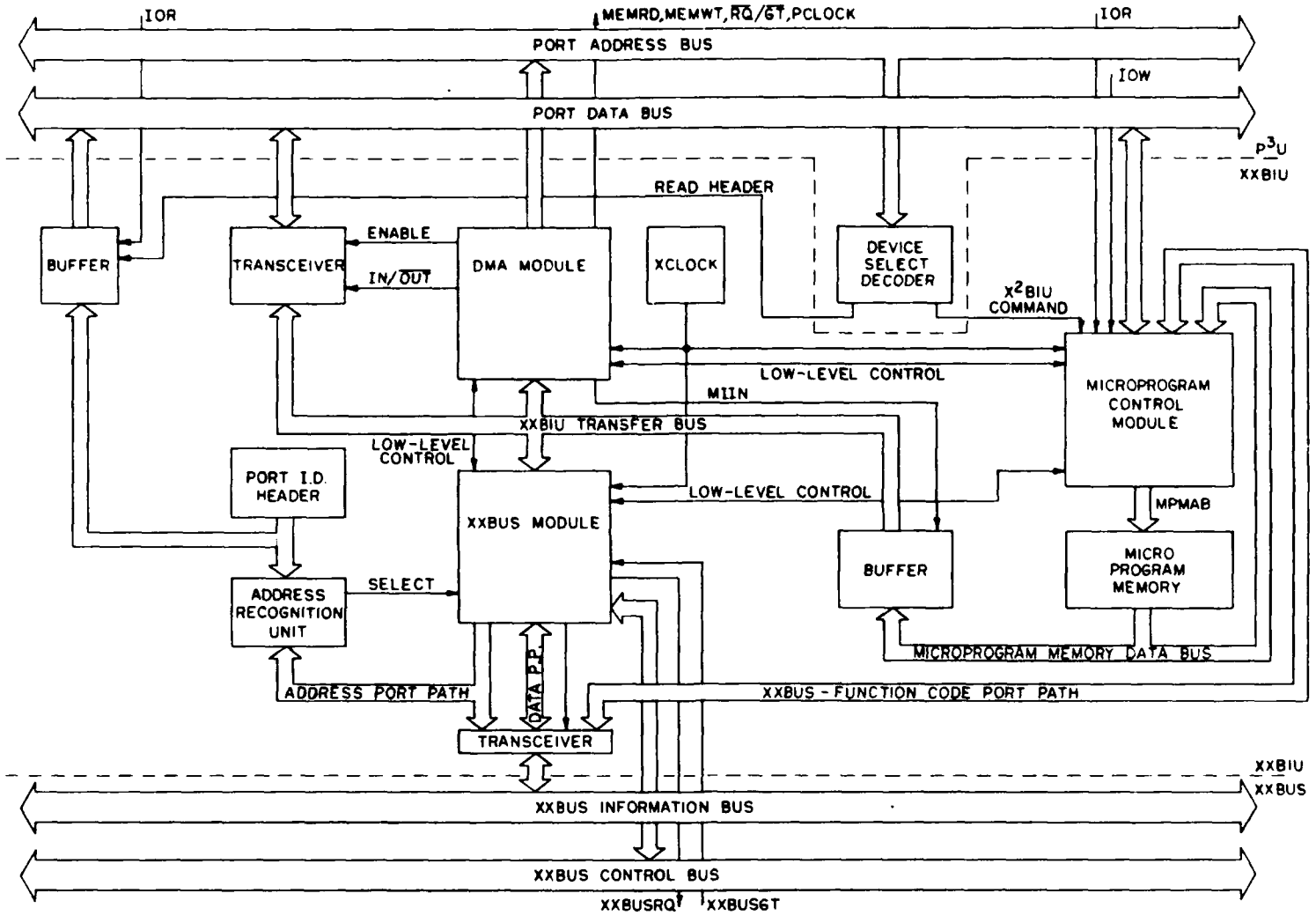


Figure IV-08. XXBUS Interface Unit Functional Block Diagram

associated data. The microinstruction word format is depicted in Figure IV-09. This method of partitioning greatly simplifies addressing logic. These three addressing levels will be controlled or handled differently by the control logic residing in the Microprogram Control Module.

Each microprogram of the XXBIU corresponds to a function that the XXBIU can be directed to perform. The function (microprogram) to be executed is specified by Function Codes which are written to the XXBIU via the Port Bus by the PPPU or via the XXBUS Function Code Bus by other Ports. Although each microprogram must be specified by an unique Function Code, Function Codes do not always specify microgrograms; that is, Function Codes are also used to specify hardware implemented functions such as Port Reset, XXBIU Reset, Enables, mode setting, and status reading.

The Microprogram Control Module (MCM) will contain the control logic and primitive hardware elements needed to control the execution of Microgramss. It will also contain the command and monitoring facilities which are used by the Primary Port Processing Unit (PPPU) and the XXBUS to control the XXBIU at the command level. The principle architectural features of the MCM involve support of two duplex channels. The MCM will be able to execute four microprograms at the same time to support the two duplex channels through the use of special multiplexing hardware. This capability supports simultaneous active and passive IPC dialogues.

The DMA Module (DMAM) will contain the hardware elements used by the XXBIU to directly access the port memory. The DMA module will support two channels of block transfers and direct addressing from the microprograms. Thus, the addressing parameters of two channels can be maintained simultaneously as the DMAM is used by the microprograms to directly address the port memory to fetch processing parameters, etc. All DMAM activities will be driven from MI directives. This arrangement makes the number, type, and mix of data transfer activities and data processing activities programmable.

The XXBUS Module (XBM) will contain the XXBUS transmitter and receiver elements along with the Conditional Logic Unit (CLU) which is used to decode

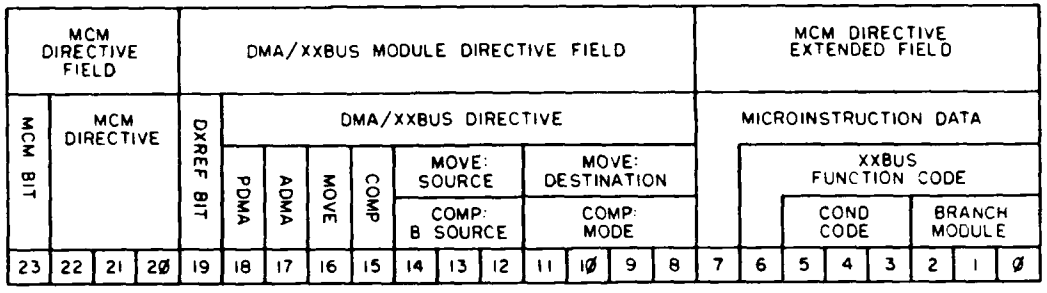


Figure IV-09. Microinstruction Word Format

control data and to perform data validity checks. Principle features of the XXBUS Module include support of two channels and the Transmit/Receive (XMIT/RCVE) control logic. Each channel will have dedicated Port addressing registers and redundant check word generators. The XXBUS receive and transmit control logic will be implemented using hardware Algorithmic State Machines (ASMs). This will minimize the XXBUS cycle time by eliminating synchronization latencies and microprogram response times.

The MI directive path, the XXBIU Transfer Bus (XTB), the XXBUS Port Paths and their associated control logic are designed for parallel operations. The MI directive path will be dedicated solely for transferring information representing MI directives. This dedication will permit the pipelining or queuing of MI directives concurrently with other XXBIU activities. It will also simplify the memory control circuitry, making memory access time the primary limiting factor in reducing MI directive queuing times. The XTB will carry data between the XXBIU and the Port Data Bus. These operations will be performed in parallel with unrelated activities involving the XXBUS and/or the MI directive path. This arrangement will also help reduce the impact of the relative slowness of these transfers on the overall throughput of the XXBIU. The XXBUS port paths will be controlled by dedicated ASMs containing transmit and receive control logic. This differs from the XTB in that the XTB will be driven directly by the MI directives and the XXBUS transmit and receive controllers will almost completely be decoupled from MI directive control.

4.4.1 Microprogram Memory. The Microprogram Memory (MM) contains the microprograms that drive the XXBIU hardware activities. The following is a description of how these microprograms are mapped into the MM and how microprogram memory addressing supports the logical flow of XXBIU operations.

4.4.1.1 Mapping. The mapping of microprograms into the MM is constrained by the addressing hardware which imposes a partitioning of the MM into three hierarchically arranged levels. These levels are microprograms, microprogram modules, and microinstructions. At the highest level, the MM is divided into

32 sections where each section is a microprogram. At the second level, each microprogram is divided into eight (8) microprogram modules. At the lowest level, each microprogram module contains 8 microinstructions.

4.4.1.2 Microprogram Memory Addressing. Addressing at each level is accomplished through three different address fields of the microprogram memory address. These three fields correspond to the partitioning of the MM and are referred to as the microprogram address, the microprogram module address, and microinstruction address. The most significant difference between these three address fields is how they are specified. The microprogram address is specified externally to the XXBIU by writing a Function Code to the XXBIU. This is the only way in which a microprogram address can be specified; that is, the XXBIU cannot by itself modify a microprogram address. The microprogram module address is specified by, and only by, the microprograms themselves as they are executed. A microprogram elects to change its microprogram module address by issuing a microinstruction that involves branching. The microinstruction address can be specified only by the XXBIU hardware. When the microprogram and/or the microprogram module address is changed, the microinstruction address will be automatically reset to zero (0). When a microprogram module begins executing, the microinstruction address will be incremented automatically each time a microinstruction is fetched.

4.4.1.3 Microprogram Logical Flow. The logical flow of XXBIU operations will be controlled at three levels which correspond to the partioning and addressing of the microprogram memory. Logical flow at the microprogram level is a process of deciding which microprogram to execute and will be controlled externally by the primary port processing unit or by other ports via the XXBUS. This is a process of deciding which Function Code to send to the XXBIU. At the second or MP module level, logical flow is a process of deciding the next MP module address and is controlled by the microprograms themselves via the Jump (JMP) MI directives and the hardware it drives. At the third or microinstruction level logical flow will be sequential and controlled by the hardware.

4.4.2 Microprogram Control Module. The Microprogram Control Module (MCM) contains the mechanisms that provide for XXBIU command and monitoring, Function Code control, microprogram control, and for microinstruction control. XXBIU command and monitoring provides the PPPU and the XXBUS with the mechanism needed to reset, enable, disable, select the operational mode of, and to read the status of the XXBIU. Function Code control involves the movements of function codes within the XXBIU. Microprogram control involves the logical flow of microprogram execution. Microinstruction control involves the fetching, decoding, and execution of microinstructions. Figure IV-10 shows the functional block diagram of the microprogram control module. This subsection will describe how these elements are organized and how they operate to support these mechanisms.

4.4.2.1 XXBIU Command and Monitoring. The elements of the MCM that provide for XXBIU command and monitoring include the Port Command Unit (PCU), the XXBUS Command Unit (XCU), and the Status Interface Unit (SIU). The following is a description of each of these units.

4.4.2.1.1 Port Command Unit (PCU). The PCU will provide the PPPU with hardware level control of the XXBIU. Through the use of the PCU, the PPPU will set the operational mode, control status reads, set hardware control flags, and load Function Codes from the port bus. These commands will be sent asynchronously to the PCU via the port data bus by using I/O write cycles, controlled by the two lines I/O write (IOW) and XXBUS command. The IOW line is the port system bus I/O write cycle timing control line and will be used to control the command decode sequenc[illegible]ing of the PCU. The XXBUS command line comes from the port bus devi[illegible] decoder and will be used to enable the PCU for I/O cycles.

4.4.2.1.2 XXBUS Command Unit (XCU). The XCU will provide other MPC Ports with hardware level control of the XXBIU. Through the use of the XCU, other MPC Ports may issue commands that consist of enabling, disabling, and resetting the XXBIU. These commands will be set asynchronously to the XCU via the XXBUS by the use of XXBUS write MI directives from the other ports.

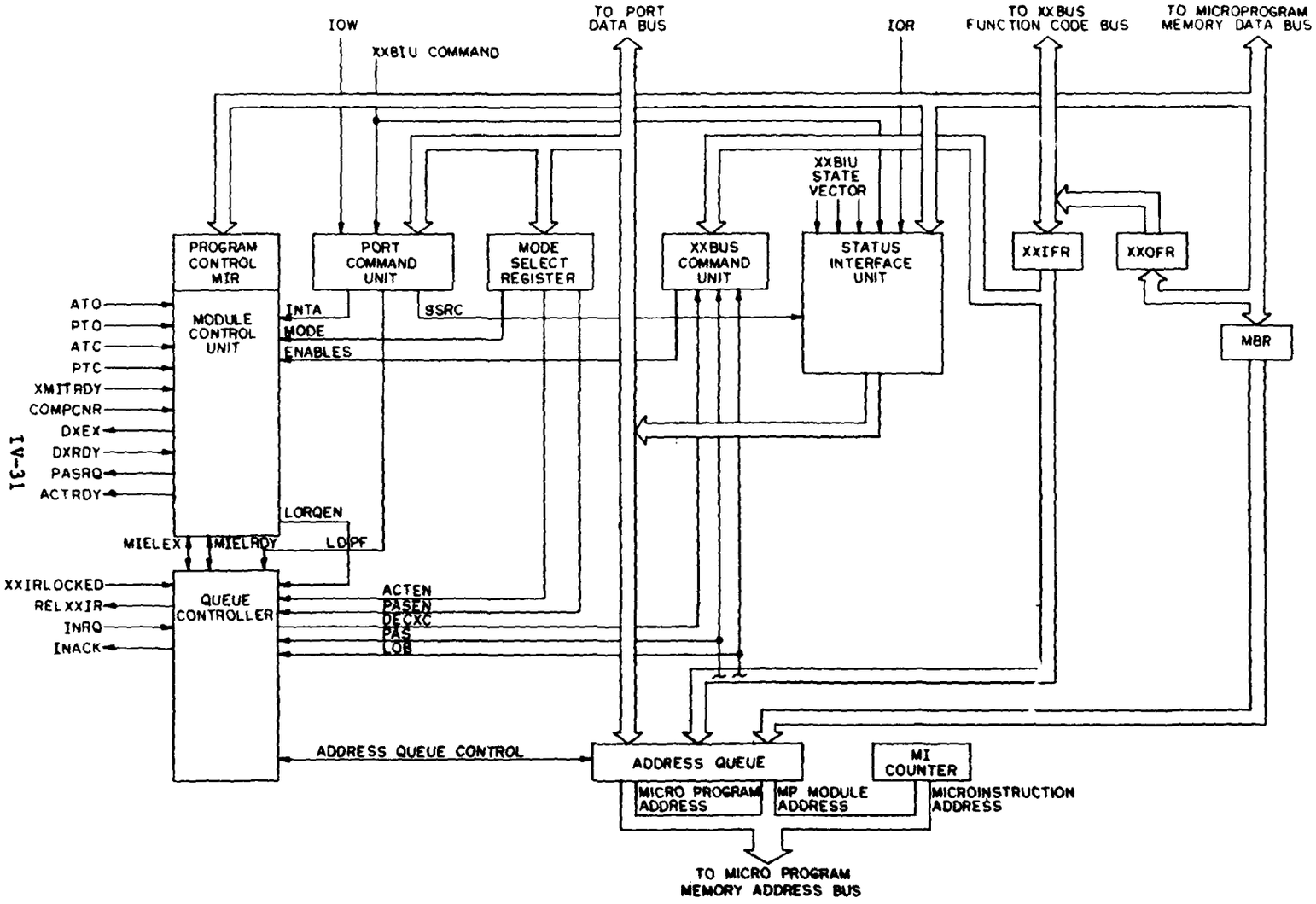


Figure IV-10. Microprogram Control Module Functional Block Diagram

The commands will be encoded in Function Codes received by the XXBIU via the XXBUS Function Code Bus and the XXBUS Input Function Code Register (XXIFR). Commands will be indicated by the setting of the Passive-Active Switch (PAS), and setting of the Lock On Request Bit (LOB). When the PAS is set to select the passive channel (PAS=0), and the LOB is set to indicate a lock on request (LOB=1), then a command is indicated and the XCU will decode and execute the specified command. The timing of this execution sequence will be controlled by the queue controller using the Decode XXBUS Control (DECXC) line.

4.4.2.1.3 Status Interface Unit (SIU). The SIU will provide the PPPU with the capability to monitor the state of the XXBIU at 2 levels. At the microprogram level, two status registers, one for both active and passive IPC, are provided. These two registers are updated from Microprogram Memory (MM) via the MM data bus when an 'update status' MI directive is executed. At the hardware level, the state of principle hardware control and status lines can be monitored. Timing and control of status reads are provided by the two control lines I/O Read, (IOR) and XXBUS command. The IOR will be the port system bus I/O read timing control line and is used to control the output sequencing of the SIU. The XXBUS command line originates from the port memory bus device select decoder and will be used to enable the SIU for I/O read cycles.

4.4.2.1.4 Interrupt Request Lines. In addition to the SIU, there are two interrupt request lines, Active Ready (ACTRDY) and Passive Request (PASRQ). The setting of these lines will be controlled via the 'set interrupt request' MI directive. Once set, each line is cleared only when the PPPU sends the appropriate 'clear interrupt request' command to the Port Command Unit (PCU). These two interrupts and the SIU can be used to implement a high level communications protocol between the microprograms and the PPPU firmware.

4.4.2.2 Function Code Control. Function Code Control involves the movements of Function Codes internally to the XXBIU. The following paragraphs describe the structure and operations of the data path and control elements that support this movement.

4.4.2.2.1 Function Code Transfer Paths. Function Codes will enter the Microprogram Control Module (MCM) from either the port data bus or the XXBUS Function Code bus. The Function Code from the port data bus will be loaded directly into the Address Queue (AQ) by the Port Command Unit (PCU), when it receives a load-execution-code command from the PPPU. Function Codes that enter the AQ from the port data bus can only be loaded into either the Active Port Microprogram Address Register (APMPAR) or the Passive Port Microprogram Address Register (PPMPAR) of the AQ (Figure IV-11). Function Codes that enter the MCM from the XXBUS are loaded into the XXBUS Input Function Code Register (XXIFR) by the receiver logic that resides in the XXBUS module. Once a Function Code is loaded into the XXIFR, it is held until the AQ controller rejects or accepts it. If the Function Code is accepted, it will be either sent to the XXBUS Command Unit (XCU), or to the AQ. Function Codes that enter the AQ from the XXBUS can only be loaded into either the Active XXBUS Microprogram Address Register (XAMPAR) or the Passive XXBUS Microprogram Address Register of the AQ. After a Function Code is loaded into an addressing register (AXMPAR), its next destination is the Microprogram Memory Address Bus (MPMAB). This transfer is made via the multiplex switch which connects all the addressing registers of the AQ to the MPMAB. This multiplex switch is controlled by logic contained in the AQ controller.

The path taken by Function Codes when leaving XXBIU will originate in Microprogram Memory (MM) and end at the XXBUS Function Code bus. An outgoing Function Code is stored in the data field of a "load XXOFR" MI directive. When this MI directive is executed, the Function Code is loaded into the XXBUS Output Function Code Register (XXOFR). The Function Code is held until the XXBUS transmitter logic, located in the XXBUS Module (XBM), gains control of the XXBUS. At that time, the Function Code is sent out onto the XXBUS and is transferred to the XXIFR of the target port.

4.4.2.2.1.1 Function Code Control - Port Data Bus. Function Codes will be loaded from the port data bus into the AQ when the PPPU I/O writes a 'load function code' command to the Port Command Unit (PCU). To support the PPPU in this operation, the MCM will provide two interrupt request lines, ACTRDY

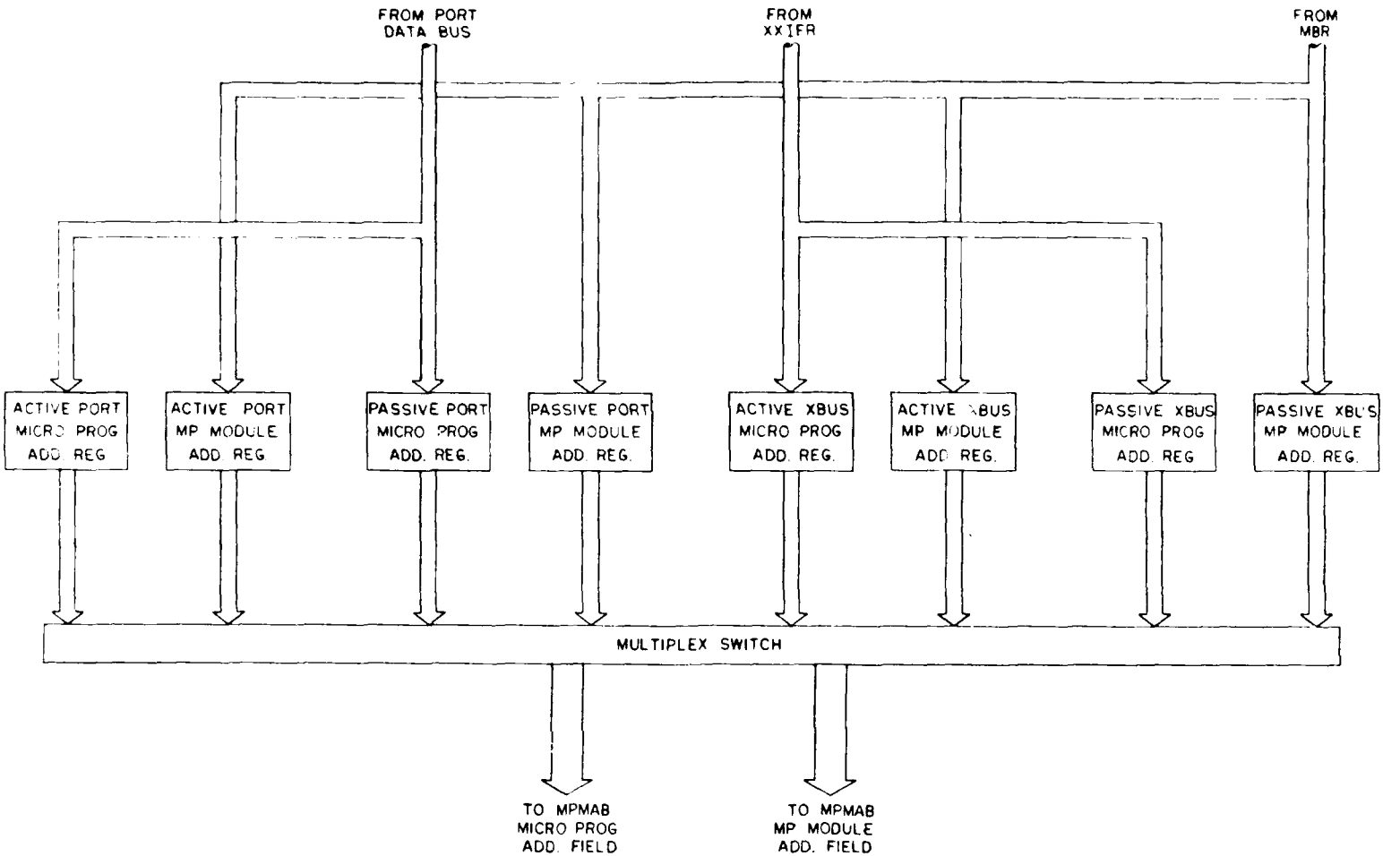


Figure IV-11. Address Queue Organization

and PASRQ, and two status bits via the Status Interface Unit (SIU). The two interrupt request lines are set to their active state from the microprograms when a "set interrupt" MI directive is executed, and are reset when the PPPU I/O writes an "acknowledge interrupt" command to the PCU. These two interrupts are used by the XXBIU to indicate when a microprogram requires servicing from the PPPU. The two status bits are connected to the execute request flags of the Active and Passive Port Microprogram Address Registers (APMPAR and PPMPAR). These two status bits thus provide the PPPU with the means to verify the availability of the APMPAR and the PPMPAR prior to loading a Function Code.

4.4.2.2.1.2 Function Code Control - XXBUS. Function Codes entering from the XXBUS Function Code bus will go through two transfer sequences before reaching the XXBUS Command Unit (XCU) or the AQ. The first transfer, from the XXBUS to the XXIFR, will be controlled by the XXBUS receiver logic located in the XXBUS Module (XBM). The second transfer, from the XXIFR to either the XCU or the AQ, will be controlled by the AQ loader located in the AQ controller. These transfers will be supported by the control status lines PAS, LOB, LORQEN, ACTEN, PASEN, EXXARQ, and EXXPRQ. During each transfer, control of the XXIFR will be maintained by the controller of that transfer. Transfer of XXIFR control will be performed using 'handshaking' techniques and control lines XXIRLOCKED, RELXXIR, INRQ, and INACK.

The following paragraphs describe the sequence of events that will take place in the XXBIU when Function Codes are received from the XXBUS.

- The XXBUS receiver will transfer the Function Code into the XXIFR. Once the XXIFR has been loaded, the XXBUS receiver will begin the control transfer sequence with the AQ loader.

  Immediately after a bus cycle, during which the XXBUS input holding registers have been written into, the XXBUS receiver logic will take INRQ to its active state and wait for INACK to assume its active state. When the AQ loader senses that INRQ is active, it will

respond immediately by taking INACK to its active state. When INACK becomes active, the XXBUS receiver logic returns INRQ to its inactive state and becomes available for the next bus cycle. When INRQ becomes inactive, the AQ loader returns INACK to its inactive state.

- o The AQ loader will then determine what action is required, regarding the Function Code just received.

- o The next event to occur, is the release of XXIFR. Once loaded, XXIFR is locked, indicated by XXIRLOCKED becoming active, and remains locked until the release sequence is executed. This sequence will be executed by the loader if the Function Code is rejected, or by the microprogram when the Release-Input Data Register (XXIDR) MI directive is executed. RELXXIR will be taken to its active state when this sequence is initiated. The locking flip-flop, located in the XBM, will be cleared and the control line XXIRLOCKED becomes inactive. When XXIRLOCKED becomes inactive, RELXXIR will be taken to its inactive state. Once the XXIDR is released, the XXBUS receiver will permit its loading during the attempt to write to it by the XXBUS.

4.4.2.2.2 Microprogram Memory Address Registers and Multiplexor. The switching of the Microprogram Memory Address Registers, (MPMAR), onto the Microprogram Memory Addressing Bus, (MPMAB), will be controlled by the multiplex logic located in the Address Queue (AQ) controller. This multiplex logic will consist of two sets of flip-flops and an ASM. Each set contains four flip flops. The first set of four flip-flops will serve as execute request flags for each of the MPMARs, with the second set of flip-flops controlling the multiplex switch. The ASM will implement the logic as a rotating, last handled, lowest priority multiplexing scheme. This scheme will be used to control the scanning and latching of requests from the execution request flag and to control the setting of the multiplex switch control flip-flops.

4.4.2.3 Microprogram Control. The Microprogram Control mechanisms support the logical flow of microprogram execution. These mechanisms include those hardware elements that implement the MI directives used by the microprograms to perform conditional and unconditional branching to one of eight microprogram modules within a microprogram, and termination/subtermination of microprogram execution. It will also include sections that provide direct hardware control of the logical flow including reset and special timer triggered activities. In all cases, microprogram module branching will be a process of selecting between alternative Microprogram Module Addresses used to address the next microprogram module. The following paragraphs describe the hardware elements and operations involving the manipulation of the microprogram module address fields of the microprogram memory address registers.

4.4.2.3.1 Microprogram Control Elements. The following describes the elements that provide for microprogram control and how they are organized. These elements include the data paths and those that control, as directed by the microprograms, the operation associated with microprogram control.

4.4.2.3.1.1 Microprogram Control Data Paths. The registers that support Microprogram Control include the Module Branch Register (MBR), and the four Microprogram Module Addressing Registers (MDAR) of the Microprogram Address Queue. The MBR will hold the if-true-branch target module address during the execution of conditional branch MI directives. Each of the MDAR's will hold the address that points to the next microprogram module of the associated microprogram to be executed.

As shown in Figures IV-10 and IV-11, circuit paths associated with these registers include:

- o the Microprogram Memory Data Bus (MPMDB)
- o the Branch Microprogram Module Bus (BMMB)
- o the multiplex switch
- o the Microprogram Memory Address Bus (MPMAB)

The MPMDB will carry the MP Module Branch Address contained in the data field of branch MI directives to the MBR during MI directive fetch cycles. The BMMB will carry the MP Module Branch Address from the MBR to any one of four MDARs. From the MDAR, MP Module Address will be connected to MPMAB by the multiplex switch.

4.4.2.3.1.2 Microprogram Control Hardware. The Microprogram Control Hardware is represented in Figure IV-10 by the Microprogram Control Module Control Unit (MCMCU). The MCMCU will contain the sequence controller that performs transfer operations along the Microprogram Control data path, hardware that decodes and executes the branch MI directives, and hardware that implements the channel time out operations.

Associated with the MCMCU will be the following control and status lines supporting the Microprogram control hardware:

- o >ATO: This line represents the timing element used to control time related activities of the active channel.
- o >PTO: Passive channel version of ATO.
- o ATC: This line provides the status of the active channel DMA block transfer word counter.
- o PTC: Passive channel version of ATC.
- o COMPCND: This line provides the result of the compare logic unit, which is contained in the XXBUS module, during the execution of compare MI directives.

With respect to the organization of these control lines, each circuit path of the microprogram control data paths will have a different ASM controlling its activities; that is, the fetch MI directive activity associated with MPMAB and the MPMDB, the MI directive execution activity associated with the BMMB, and multiplex switch control will all be controlled by separate, yet, coordinated hardware logic.

4.4.2.3.2 Microprogram Control Operations. The operations involved with the control of microprogram logical flow will be processes that determine the next state of the Microprogram Module Address Registers (MDAR). These operations consist of two types. The first type provides microprogram controlled logical flow or branching. This control is represented by the MI directive set that includes: DMA Block transfer branch activities, compare logic branch activities, unconditional branching, and microprogram termination/subtermination. The second type of activity provides for direct hardware control of microprogram logical flow.

4.4.2.3.2.1 DMA Block Transfer Branch Operations. DMA block transfer operations will be performed by the DMA Module (DMAM) under direction of the microprograms. The MI directives to be sent to the DMAM include those that set up the DMAM addressing parameters and those that direct the DMAM as to when a word transfer can take place with respect to other possible MI directive activities. This direct control by the microprograms will be performed by the execution of the DMA tranfer MI directive and the JMP ATC/PTC MI directive set.

The block transfer microprograms of the two DMA channels will be implemented using the appropriate versions of these two MI directives in a conditional loop structure. During the execution of these microprograms, the DMAM will transfer a single word of the block to be transferred each time it receives a DMA transfer MI directive. The JMP (ATC/PTC) MI directive will be used to determine when all words in the block have been transferred.

The JMP (ATC/PTC) MI directives will be supported by the status lines ATC and PTC which indicates to the Microprogram Control Module (MCM) the word count status of their respective channel's word counter. Each time a DMA transfer MI directive of a channel is executed, that channel's word counter will be decremented by one. When the channel's word counter reaches its terminal count state, the terminal count status line, ATC/PTC, of that channel will become active. Once it becomes active, the execution of that channel's JMP-IF-ATC/PTC MI directive will result in the microprogram module jumping to the microprogram module specified by the MI directive, ending the

block transfer operation. Also, when the terminal count condition for a channel is reached, the execution of any subsequent DMA transfer MI directives by the DMA Module will be inhibited. This action is required because of the possible pipelining of the DMA transfer MI directives.

Any number or mix of MI directives can be contained within a DMA transfer microprogram module, except those that might erroneously change the DMA parameters. These additional MI directives may be included to perform integrity checking, data comparisons, or subterminations. The inclusion of subterminate MI directive will permit the multiplexing of a DMA operation with the operations of any other microprogram.

4.4.2.3.2.2 Compare Logic Unit (CLU) Branch Operations. The XXBIU can be programmed to perform any one of a number of compare operations on data that is received from the XXBUS. The hardware that performs these operations, as directed by microprograms, resides within the XXBUS Module and are referred to collectively as the Compare Logic Unit (CLU). A compare operation will begin when a Compare-Jump MI directive is received by the XXBUS Module (XBM). During the fetch of this directive, the Branch Module Address will be loaded into the MBR. During the execution of this directive, the CLU will indicate the results to the MCMCU via the CMPCND status line. If the compare tests true, then CMPCND becomes active. CMPCND becoming active will result in the target MP Module address being loaded into the Microprograms MDAR. If the compare does not test true, the execution will terminate. This communication arrangement requires that the MCMCU and XBM remain synchronized throughout the entire execution of Compare-Jump MI directive.

4.4.2.3.2.3 Unconditional Branching. This operation will be performed when an unconditional Jump MI directive is received. During the execution of this MI directive, the Module Address which was loaded into the Module Branch Register (MBR) during the MI directive Fetch cycle will be loaded into Module Address Register (MDAR) of the executing microprogram. Also during this execution cycle, the microinstruction counter will be set to zero. Thus, the next MI directive to be executed after the execution of an unconditional Jump will be the first MI directive of the target microprogram module.

4.4.2.3.2.4 Termination/Subtermination MI Directive. The termination MI directive will be used by the microprogram to indicate to the hardware that it has finished, thus terminating its execution. If this MI directive is executed by a portside microprogram, the MCM will clear the appropriate EXRQ flag in the multiplex logic and will also set the availability status flag to indicate to the PPPU that the respective port channel is available for the next XXBIU operation. If this MI directive is executed by an XXBUS-side microprogram, then the MCM will clear the appropriate EXRQ flag in the multiplex logic and will indicate to the loader logic that the channel is ready to receive the next microprogram Function Code from the XXBUS. In either case, the microprogram module address register will be set to zero and the Multiplexor will advance to the next requesting microprogram.

The operation of subtermination will be provided to implement the microprogram controlled multiplexing scheme. When a microprogram completes executing MI directives that require, for reasons of integrity and speed, the microprogram to maintain control of the XXBIU, the microprogram will issue a subterminate MI directive, which will result in the multiplexor advancing to the next requesting microprogram. When the MCM executes this instruction, the module address that was loaded into the MBR during the fetch sequence, will be loaded into the MDAR of the subterminating microprogram. After the MDAR is loaded with this "return" address, the MCM will advance the multiplexor and reset the microinstruction counter to zero.

4.4.2.3.2.5 Channel Time Out Activity. Channel time out will be provided for the synchronization of microprogram level communication between XXBIUs by supporting the request/respond-or-time-out protocol used to achieve this synchronization. The time out activity during this synchronization process will be as follows:

- The microprogram of the initiating port assembles the appropriate request information packet and sends it out onto the XXBUS.
- The requesting microprogram then starts the appropriate timer.

- o After starting the timer, the microprogram issues a terminate MI directive and waits for a response Function Code or for the time out condition. If the Function Code is received, the microprogram that is subsequently executed stops the timer. At this point, the microprograms are synchronized. If the time out condition occurs, then the MCU will issue an interrupt to the port processor.

4.4.2.4 Microinstruction Control. The XXBIU Microinstruction set supports the activities needed for the three modules encompassed in the XXBIU. These include the XXBUS, DMA, and Microprogram Control Modules. Each module can operate independently, dependently, synchronously, or asynchronously on the portion of the MI directive that they receive and operate on. In support of this operating environment, the MI directive set is functionally partitioned into two types of directives: XXBUS/DMA Control, and MCM Control. One type of directive need not be related to the directive specified in the other type. This partitioning will provide for increased throughput by eliminiating the need for synchronization of the entire XXBIU for each MI directive, while supporting the concurrent operation of each module.

The operations asociated with each MI directive are further divided into two phases: fetch and execution. MI directive fetch phase relates to the process of transferring the MI directive from the microprogram memory (MM), to the execution hardware of the XXBIU. The execution phase coordinates the hardware primitives that execute the MI directives specified with each microinstruction.

There are four possible combinations of MI directives, each requiring different fetch and execution cycles. These combinations include the XXBUS/DMA directive, MCM directive, a combination of both that are executed independently, and a combination of both where one requires the coordination of the other. The MI directive, as well as the state of the XXBIU, both contribute to the number of fetch and execution cycles required.

4.4.2.4.1 MI Directive Fetch Phase. This phase includes the execution of four sequential processes: determination of when the fetch cycle should take place, analyzing the MI directive, the XXBIU status to determine which modules are involved, synchronization of the modules involved, and loading of the MI directive into the specified module's Microinstruction Holding Register (MIHR).

The determiniation of when to fetch will be controlled by the multiplex logic, the Microprogram Control Module (MCM), and the Micronistruction Execution Logic (MIEL). The multiplexor and the fetching logic will coordinate this process by handshaking the lines MIELRDY and MIELEX. MIELRDY indicates to the multiplexor that the MIEL is ready to begin execution of the next microprogram. MIELEX is used by the multiplexor to indicate to the MIEL tha the next microprogram is ready to be executed.

The determination of the modules involved and the sequencing required is reflected by the setting of the key control bits in the microinstruction word, and the status of the Transmit Ready (XMITRDY) and Compare Condition (CMPCND) control lines. Figure IV-09 showed the format of the XXBIU microinstruction. Of the bits shown, 23 and 19 will be used to indicate to the fetch logic which MI directive is to be executed, along with those that specify the XXBUS output register transfer, or the Compare/Jump directive. During the early phase of the microinstruction fetch cycle, these bits are to be tested by the fetch logic and their setting determines the nature of the remaining cycle.

If the XXBUS/DMA module directive of the microinstruction is to be executed, the fetch logic will use the handshake lines DXRDY and DXEX to synchronize these modules with the fetch operation. If only the MCM portion of the MI directive is to be executed, the fetch logic will wait for a ready indication from the MCM. In this case, the MI directive execution logic, DXRDY and DXEX, will not be used. When both are specified, the fetch logic will coordinate the two activities by waiting for both to become ready before execution of the load sequence.

When the loading of the MI directive into the XXBUS/DMA MIHR is required, and either the XMITRDY line is not active, or if the XMITRDY is active and the MI directive does not involve the XXBUS output register, the DXRDY line will be checked. The DXRDY line will be set to an active state by the XXBUS/DMA module when it is ready to receive the next MI directive. When this occurs, the MI directive data bus will be gated to the MIHR's of both modules. When the fetch logic retrieves an MI directive with bit 19 set and the DXRDY line active, DXEX will be activated. When this occurs, the XXBUS/DMA modules will respond by deactivating the DXRDY line. This action latches the MI directive into the XXBUS/DMA MIHR's and starts the execution phase when the DXRDY becomes inactive. The fetch logic deactivates the DXEX line and proceeds with the next fetch sequence, if the Compare/Jump portion of the MI directive was not specified. If the Compare/Jump is specified, the fetch logic will wait for either the CMPCND or the DXRDY line to become active before proceeding with the next fetch operation.

When loading of an MI directive into the MCM MIHR is required, and either the XMITRDY is not active, or the XMITRDY is active and an XXBUS output register is not part of the MI directive, the following will occur. During the execution of the MCM portion of an MI directive, a point will be reached when the address of the next MI directive becomes valid. At this point, the fetch logic will gate the MI directive into the holding registers. When the execution of the previous MI directive is completed, the fetch logic will latch the MI directive into the holding registers, start the execution logic, and proceed with the next fetch sequence if the Compare/Jump portion of the MI directive was not specified. If the Compare/Jump was specified, the fetch logic will wait until either the DXRDY or CMPCND control lines become active before proceeding with the next fetch.

The loading of a MI directive into the MIHR of the DMA/XXBUS module, or the MIHR of the MCM when XMITRDY is not active and the MI directive involves an XXBUS output register, will be held in a wait state until the XXBUS transmitter completes the XXBUS transfer. Completion will be indicated by XMITRDY becoming active. To prevent hang up of the microprogram by the transmitter, a time out condition can also terminate this wait state.

4.4.2.4.2 MI Directive Execution Phase. The MIEL will determine when to perform a MI Directive execution cycle, the hardware primitives involved in the execution of the MI directive, and the actual execution of the MI directive by the hardware primitives. The details of each of these processes is the function of the XXBIU modules involved. For the MCM, it will be a function of the MI directive being executed. The MCM execution hardware will also respond to the hardware directives of time out and reset and to the state of XXBIU. For the MCM execution hardware, all of these processes will require coordination with the fetching logic, the microprograms, and with the execution hardware of the XXBUS and DMA modules.

The determination of when to perform an execution cycle will be controlled by the fetching logic and the microprograms. The fetching logic will coordinate the MI directive loading operation with the initiation of the execution sequence. The microprograms will indicate to the MIEL when to stop execution by issuing a terminate/subterminate MI directive. The execution of either MI directive results in the MIEL transitioning to its idle state. The MIEL will remain in its idle state until it is restarted by the fetching logic.

Execution sequence of a MI directive will be a function of the operational specifics of the MI directive and the state of the XXBIU. The MI directive specified operations of conditional branching, unconditional branching, flag setting, flag testing, register transferring, and NOP will require different sequencing. This determination will be made early in the execution cycle when the MI directive is decoded. The state of the XXBIU will include the setting of the mode register, and the state of the status lines XMITRDY and COMPCND which respectively coordinate the activities of the transmitter and the CLU with the MIEL.

4.4.3 XXBUS Module. The design of the XXBUS Module (XBM) includes the hardware elements to perform activities associated with transferring information to and from the XXBUS. These elements involve transmitter and receiving XXBUS Information Packets, decoding (testing) XXBUS data, and supporting data error detection.

4.4.3.1 XXBUS Transmitter Elements. Figure IV-12 shows the functional elements of the XBM. The XXBUS Transmitting Elements (XT), include the XXBUS Output Data Register, (XXODR), XXBUS Passive Channel Output Addressing Register, (XXPOAR), XXBUS Active Channel Output Addressing Register (XXAOAR), and the transmit control logic.

The XXODR will be a 16-bit register with its input connected to the XTB and its output connected to the XXBUS data bus. Loading of the XXODR will be controlled by the XBM Control Unit (XBMCU) and is performed when the XBMCU receives a load-XXODR MI directive. Transfers from the XXODR will be controlled by the transmit control logic. The XXODR serves to decouple the activities of the XXBIU from those of the Bus.

The XXPOAR and XXAOAR registers will be 14-bits in width and are to be loaded from the XXBIU Transfer Bus (XTB) by the XBMCU when the XBMCU receives a Load-XXPOAR or Load-XXAOAR MI directive. These registers will hold the addressing information to control the routing of information packets through the XXBUS network. The XXPOAR and XXAOAR will be dedicated to the Passive and Active DMA channels of the XXBIU respectively. By having dedicated address registers, multiplexing of the active and passive DMA channels can take place without requiring the reloading of the address register each time a channel resumes operation.

The transmitter control logic will control the transfer of Information Packets from the XXBUS output registers, (XXODR, XXPOAR or XXAOAR, and XXOER), to the XXBUS. The transmitter will be an Algorithmic State Machine and coordinates its activities with the XBMCU, using lines XMITSTART and XMITRDY. Control of the XXBUS will be transferred to the transmitter using lines XBUSRQ/XBUSGT. Synchronization of the transmitter with the receiver sections of the other port on the XXBUS will be accomplished using XXBUS control lines XBCA, XBACK and XIRL. XBCA will be activated by the transmitter to indicate the start of the bus cycle on the XXBUS. XBACK will be activated by the receiver to indicate that the XXBUS Information Packet has been loaded. XBIRL will be activated by the receiver to indicate that the XXBUS Information Packet was locked out.

4.4.3.2 XXBUS Receiver Elements. The XBM Receiver Elements (XR) will include the XXBUS Input Data Register (XXIDR) and the receiver control logic which is internal to the XBM (see Figure IV-12). The XR will also include two external components: the address recognition unit (see Figure IV-08) and the XXIFR, which is contained in the MCM (see Figure IV-10).

The XXIDR will be a 16-bit register with its input connected to the XXBUS data bus and its output connected to the XXBIU Transmit Bus (XTB). The XXIDR serves to decouple the activities of the XXBUS and those of the XTB, by providing temporary storage of data received from the XXBUS during a bus cycle. Loading of the XXIDR will be controlled by the XR. Transfer of data from the XXIDR will be controlled by the Microprogram Control Module.

The Address Recognition Unit (ARU) shown in Figure IV-08, will perform the function of comparing the port's XXBUS and port ID with the XXBUS address bus. Thus, making the status line SELECT active when they are equal. The ARU will consist of a word comparator with one argument connected to the Port and Bus ID and with the other argument connected to the XXBUS address bus via the XXBUS address port path and transceivers. The SELECT status line will connect to the receiver logic located in the XBM and is used in conjunction with the XBCA to indicate to the receiver port when to engage in an XXBUS cycle.

The XR control logic will control the transfer of Information Packets from the XXBUS to the XXBUS input registers, XXIDR and XXIFR. The XR control logic will be implemented as an algorithmic state machine which must communicate with the XT logic of other ports and with the Loader Logic of the MP Address Queue Controller. Communication with the XT control logic of other ports will be supported by the bus control lines XBCA, XBACK, and XBIRL. Communication with the Loader Logic will be supported by the handshaking lines INRQ, and INACK, the status line XXIRLOCKED, and the control line RELXXIR.

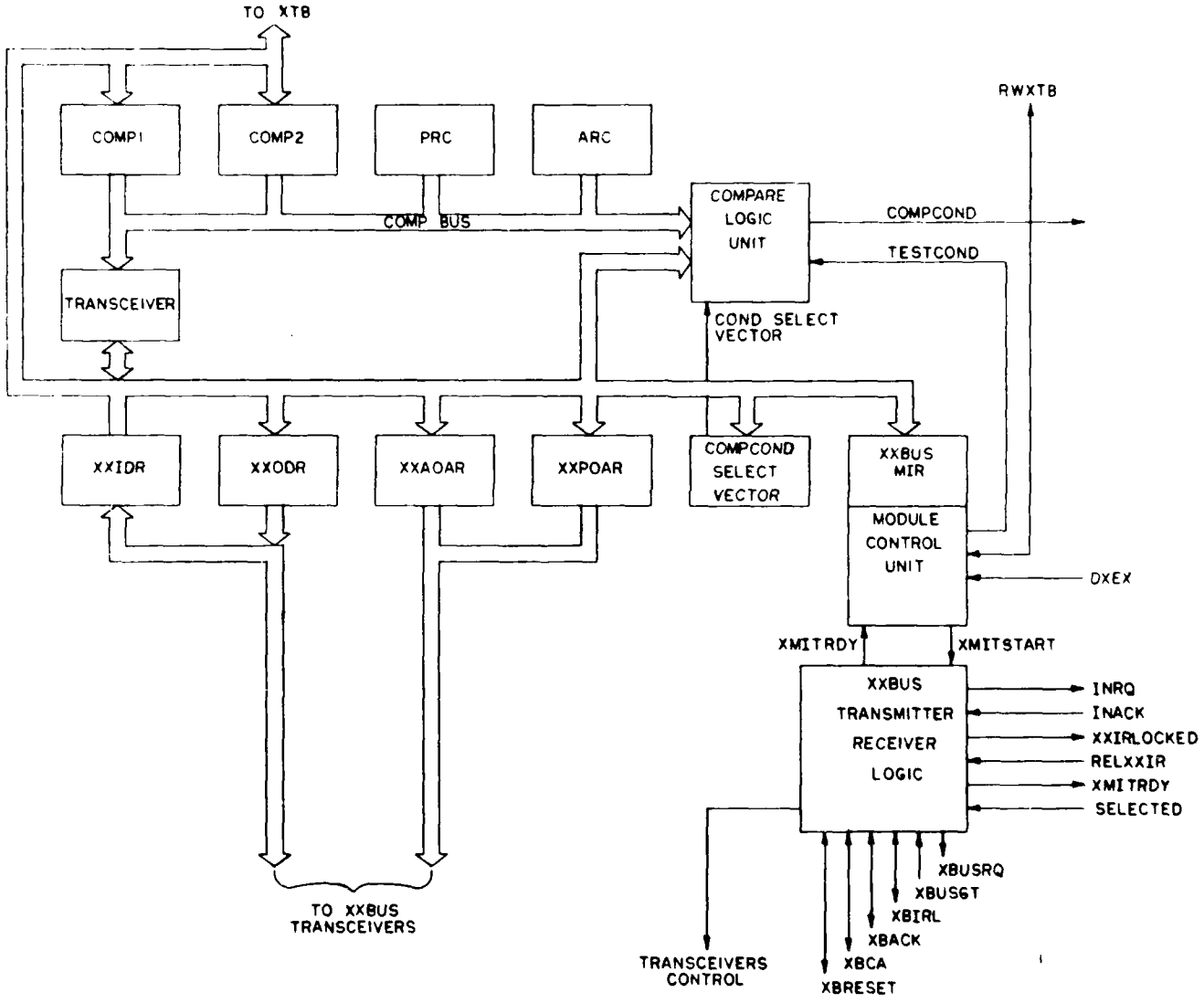


Figure IV-12. XXBUS Module Functional Block Diagram

AD-A110 858 PRC GOVERNMENT INFORMATION SYSTEMS MCLEAN VA F/G 9/2
IMPROVED MICROPROCESSOR DESIGN.(U)
OCT 81 L E KLET, R E CRANDALL, J H GLENDE F30602-80-C-0171
UNCLASSIFIED PRC-R-3412 RADC-TR-81-273 NL

2 OF 2
AD A110858

END
DATE FILMED
3-82
DTIC

4.4.3.3 Data Decoding Elements. XBM Data Decoding Elements (DDE) support the actiities of XXBUS data decoding and testing and are shown in Figure IV-12. The DDE includes CMPR1, CMPR2, XXIDR, the compare select vector, and the Compare Logic Unit (CLU). Also associated with these operations is the XBMCU which will control the DDE during the execution of the COMPARE-JUMP MI directives.

CMPR1 and CMPR2 will be 16-bit registers used to hold data to be compared with the contents of the XXIDR. This loading activity will be performed by the XBMCU when a COMPARE-JUMP MI Directive, specifying CMPR1/CMPR2, is executed. The Compare Logic Unit (CLU) will provide the various bit test, bit compare, and word compare functions required to support the data decode and test operations. The bit test operations can only be performed using the XXIDR. The compare operations will be performed between the XXIDR and either COMP1, COMP2, Active Redundant Checkword generator (ARC), or Passive Redundant Checkword generator (PRC). The type of compare/bit test will be specified by the compare select vector. The result of these compares and bit tests will be provided via the COMPCND status line. Synchronization of the CLU with the Microprogram Control Module branch control logic will be provided by the control line TESTCOND.

The COMPCND select vector register will be used to hold the COND select vector during execution of the Compare/Jump MI directive. This value is stored in the DMA/XXBUS Directive "Comp: Mode" field of Compare-JMP Microinstructions and will be loaded into the COMPCND select vector register during the MI Directive load sequence (see Figure IV-09).

4.4.3.4 XXBUS Data Transfer Error Detection. The XXBUS data transfer error detection elements (Figure IV-12), include the Passive Redundant Checkword Generator (PRCG), Active Redundant Checkword Generator (ARC) and the Compare Logic Unit (CLU).

The PRCG and the ARC will be used to generate redundant check words, and at the end of block data transfers to detect errors that might occur during the data transfer process. Each of these sections will consist of 16 toggle

type flip-flops. The inputs and outputs of each one of these flip-flops in the PRCG and the ARC will be connected via a bidirectional gate to one of the 16 lines of the COMP bus. These connections are such that the PRCG and the ARC each will have one flip-flop for monitoring each of the 16 lines of this bus.

The CLU will be used to compare the check word of its port (read from the PRCG or the ARC via the COMP bus) and the XXIDR (which will hold the check word received from the other port involved in the data transfer).

The XBMCU will control the execution of the error checking MI directive. Data transfer processes using error checking consists of four phases: initialization, data transfer, check word transfer, and check word compare. Before data is transferred between the XXBIU, the PRCG or the ARC, depending on the DMA channel involved, must be set to their initial condition (0).

After the PRCG or ARC has been initialized, the data transfer will begin. During the transfer operation, data streaming through the XTB will also be gated onto the COMP bus where it will be looked at by the PRCG or ARC. The flip-flops monitoring the COMP bus will change state each time they see a high in one of the 16 bits. Since the PRCG or ARC of the XXBIUs see the same data stream, the check word generated by each XXBIUs PRCG or ARC should be equal. At the end of the data transfer, one of the ports sends its check word to the other. The port receiving the check word places it in its XXIDR and proceeds to compare it with the check word generated by its own PRCG or ARC.

4.4.3.5 XXBUS Transmitter Operation. The XT and XR engage in one of two possible bus cycles which correspond to the locked and unlocked states of the XXIDRs. Figures IV-13 and IV-14 show the relative timing of principle XT control lines during each of these bus cycles. Of the signals shown, Transmit Ready/Transmit Start (XMITRDY/XMITSTART) will be used to synchronize the Module Control Unit with the XTE, Exchange Bus Request/Exchange Bus Grant (XXBUSRQ/XXBUSGT) used by the XT to gain control of the XXBUS, and XBCA/XBACK/XBIRL will be used to synchronize the XR with the XT during the

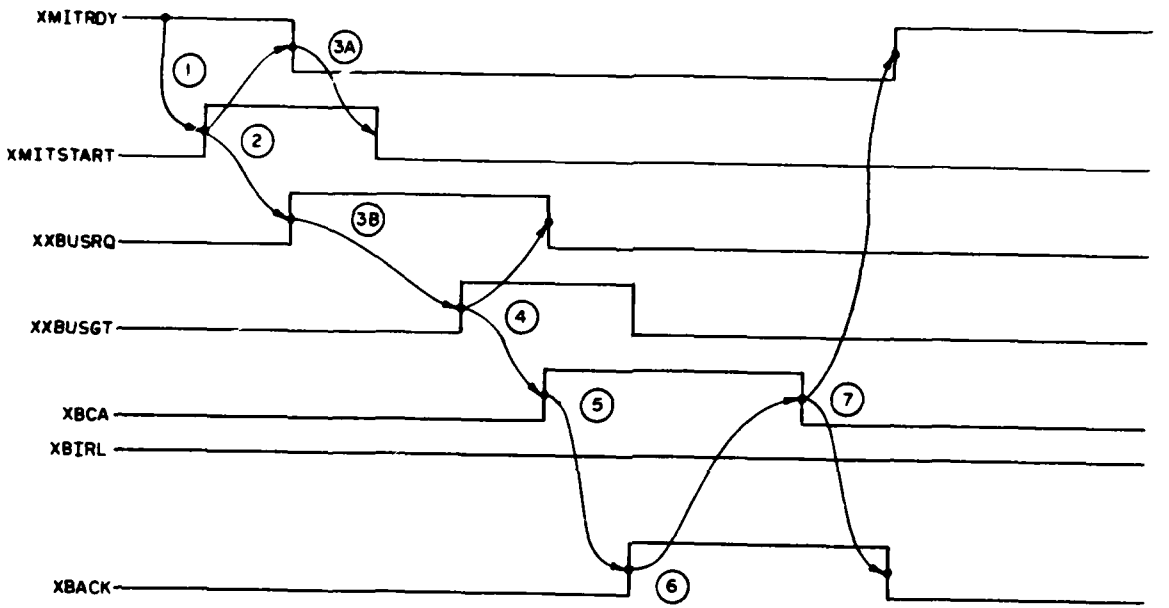


Figure IV-13. XXBUS Transmitter Timing, XXIR Unlocked

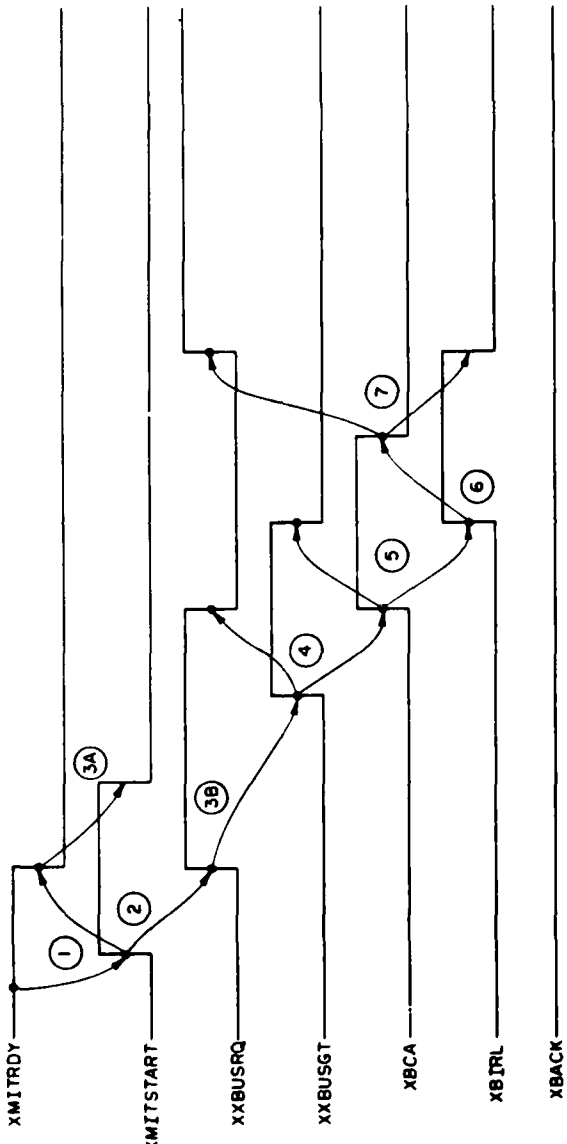


Figure IV-14. XXBUS Transmitter Timing, XXIR Locked

bus cycle. The following describes the operation of the XT with respect to the timing of these signals during each of these bus cycles:

4.4.3.5.1 XXIR UNLOCKED. Start-up of the XXBUS transmitter will be performed by the XXBUS Module Control Unit (XBMCU) after it executes a Load-XXODR MI directive. Before executing this instruction as shown in Figure IV-13, the XBMCU will test the state of XMITRDY. If XMITRDY is not active, the XBMCU will wait until it becomes active before executing the MI directive. If XMITRDY is active, then the XBMCU will execute the instruction and proceed with the transmitter startup sequence (see (1) in diagram).

The start-up sequence begins with the XBMCU making XMITSTART active (2). This action results in the transmitter leaving its ready state and moving to its XXBUS Request state. When the transmitter leaves its ready state (3), XMITRDY becomes inactive. This action will be sensed by the XBMCU which will then know that the transmitter has started operating. The XBMCU will then make XMITSTART inactive and proceed to become available for the next MI directive. When the transmitter enters its XXBUS request state (3B), it will make XXBUSRQ active. XXBUSRQ becoming active will be sensed by the XXBUS control port (Section 4.3). When the XXBUS control port decides to transfer XXBUS control to a requesting transmitter, it will do so by making the XXBUSGT signal to that transmitter active. When XXBUSGT becomes active (4), the transmitter will leave its requesting state which results in the deactivation of XXBUSRQ and the beginning of the actual bus cycle sequence.

The bus cycle will begin with the transmitter making XXBCA active (5). At the same time, the contents of the XXBUS output registers are placed onto the XXBUS. When XXBCA becomes active, all receivers on the XXBUS will look at the XXBUS address and determine whether or not to accept the bus cycle. Also, the XXBUS Control Port will know that the transmitter started and will proceed to deactivate XXBUSGT and to queue up the next grant. The receiver accepting the bus cycle will proceed to load the XXBUS information into its Input Registers. After the XXBUS information has been loaded into the XXIRs by the accepting receiver (6), that receiver will acknowledge the transmitting port by making XBACK active. When XBACK becomes active, the

transmitter will deactivate XXBCA and release the XXBUS. When XXBCA becomes inactive (7), the receiver will proceed to deactivate XBACK. At this point, the bus cycle is completed, the transmitter will return to its ready state and the bus control port will proceed to transfer XXBUS control to the next requesting port.

Note: During this process, if none of the receivers on the XXBUS accept the bus cycle, then the XXBUS Control Port will time out and proceed to release the transmitter by supplying the proper XBACK sequence.

4.4.3.5.2 XXIR LOCKED. The start-up and XXBUSRQ/GT sequences of the XT will be the same for both the XXIR LOCKED and XXIR UNLOCKED cases. Also, as in the UNLOCKED case, the bus cycle proceeds in the same way until the accepting receiver with its Input Registers (XXIR) locked accepts the bus cycle (1-5 of Figure IV-14). However, the accepting receiver will, instead of loading the XXIR and making XBACK active (6), acknowledge the transmitting port by making XBIRL active. When XBIRL becomes active (7), the transmitter will know that the receiver accepted the bus cycle, however, did not load the XXBUS Information Packet into its XXIRS; the transmitter will then deactivate XBCA and return to its requesting state to try again. This process will be repeated until the XXIR becomes unlocked, or until the transmitter is reset by the XXBIU.

4.4.3.6 XXBUS Receiver Operations. Figures IV-15 and IV-16 show the relative timing of the principle control signals of the XXBUS receiver. Of the signals shown, XBCA, XBIRL, and XBAC will be used to synchronize the transmitter of the transmitting port with the receiver of the receiving port; In Request/In Acknowledge (INRQ/INACK) will be used to transfer control of the XXIRs from the receiver to the loader logic located in the MCM; SELECT will be a status line from the address recognition unit indicating when a receiver has been selected for a bus cycle; XXIR STROBE will be the clocking pulse used to load the XXIRs; and XXIRLOCKED will indicate the status of the XXIRs. The following describes the operation of the XXBUS receiver with respect to the timing of these signals during XXIR LOCKED and XXIR UNLOCKED bus cycles.

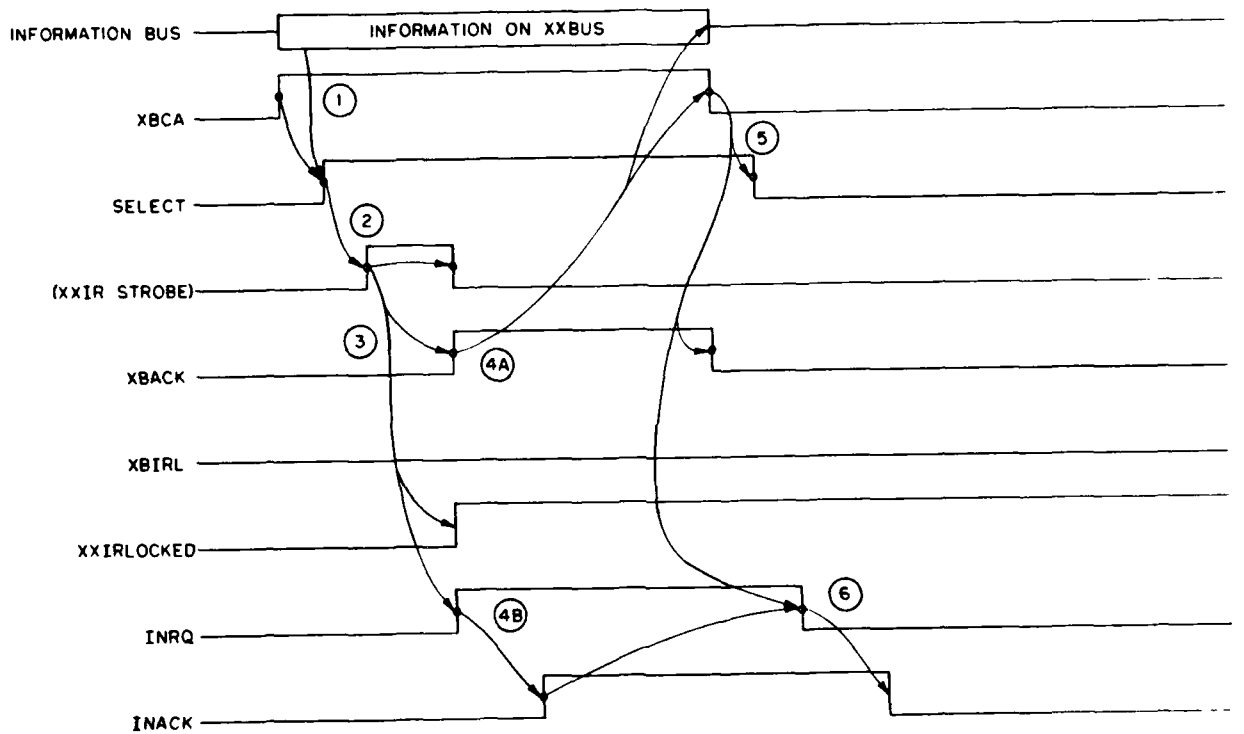


Figure IV-15. XXBUS Receiver Timing, XXIR Unlocked

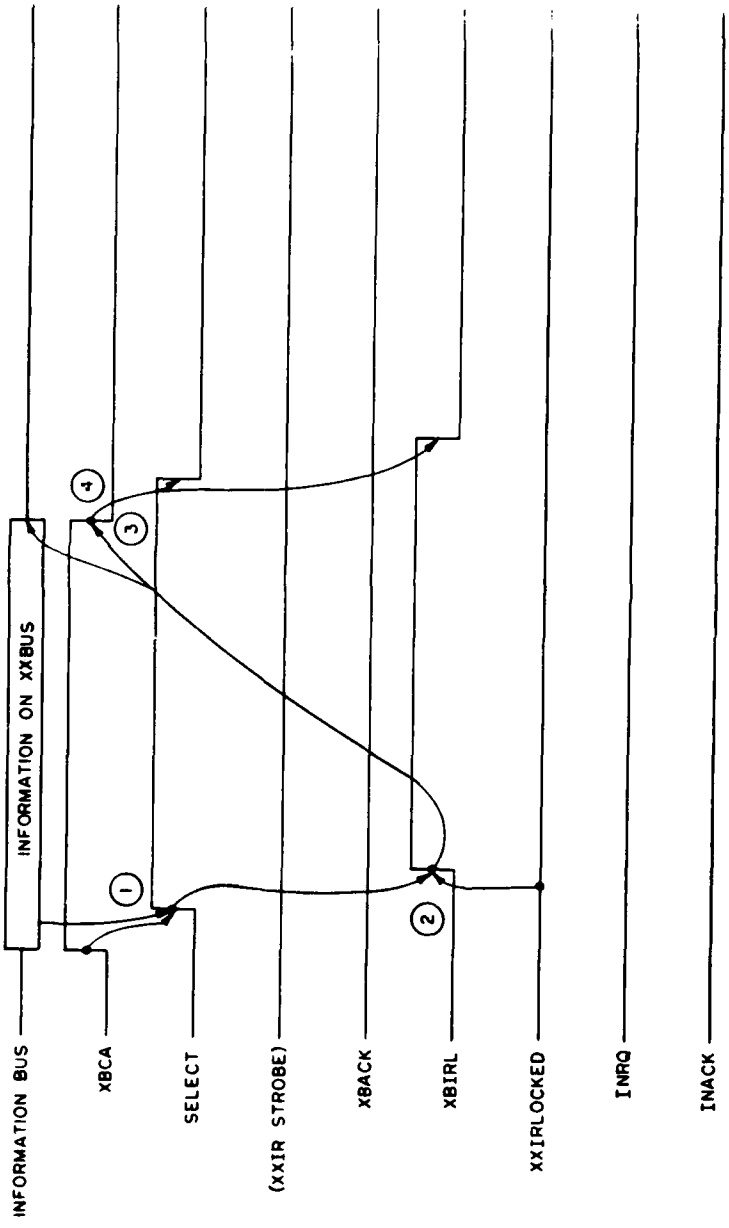


Figure IV-16. XXBUS Receiver Timing, XXIR Locked

4.4.3.6.1 XXIR UNLOCKED. The bus cycle begins when XBCA is made active by the transmitting port ((1) of Figure IV-15). When XBCA becomes active, the receiver's Address Recognition Unit (ARU) will look at the address field of the XXBUS. If the ARU decides to accept the bus cycle, it will make SELECT active. When becoming active (2), SELECT will cause the receiver to generate an XXIR STROBE pulse which will result in the XXBUS information being strobed into the XXIRs.

After loading the XXIRs (3), the receiver will acknowledge the transmitter by making XBACK active, set XXIR LOCKED active (until reset by the MCM), and begin the XXIR control transfer sequence by making INRQ active.

When XBACK becomes active (4A), the transmitter in the transmitting port will deactive XBCA.

When INRQ becomes active (4B), the loader logic will acknowledge the request by making INACK active.

When XBCA becomes inactive (5), SELECT becomes inactive, the receiver deactivates XBACK which terminates the bus cycle, and waits for INACK from the loader logic to become active.

At the completion of the bus cycle (6), the receiver will wait for the loader logic to accept control of the XXIR (indicated by INACK becoming active). When INACK becomes active, the receiver will deactivate INRQ and become available for the next bus cycle. After this time, control of the XXIR remains with the MCM until the MCM releases it.

4.4.3.6.2 XXIR LOCKED. Bus cycle initiation is the same for XXIR LOCKED as it was for XXIR UNLOCKED (see Figure IV-16). In the locked case (2) however, when SELECT becomes active, the receiver will, instead of loading the XXBUS Information Packet into the XXIR, indicate the lock condition to the transmitter of the transmitting port by making XBIRL active. When XBIRL becomes active (3), the transmitter will deactivate XBCA.

When XBCA becomes inactive (4), the ARU will deactivate SELECT and the receiver deactivates XBIRL; the receiver then becomes available for the next bus cycle.

4.4.4 XXBIU DMA Module. The XXBIU DMA Module (DMAM) contains the hardware elements that perform Direct Port Memory Access. Capabilities will include the support of concurrent DMA block transfers of two DMA channels, in addition to supporting single word access. Each of the three capabilities will have a dedicated set of address registers. DMAM control will be distributed among three algorithmic state machines which provide mechanisms for port memory bus control, coordination of processes between the DMAM and MCM modules, and controlling the decoding and execution sequencing of the DMAM MI directives.

4.4.4.1 DMAM Block Transfer Addressing. Figure IV-17, shows the functional block diagram of the DMAM. Of the elements shown, the DMAM block transfer addressing elements include the Active/Passive Page Pointers, the ADMA/PDMA Address Pointers, and the ADMA/PDMA Word Counters.

Both DMAM block transfer channels will have dedicated page address pointer registers which include the Active Page Pointer Register (APPR), and a Passive Page Pointer Register (PPPR). These page pointers will be 4-bit registers with their outputs mapped into the most significant bits of the port memory address bus and their inputs mapped into the most significant bits of the XTB. Prior to a DMA block transfer process by a channel, the page pointer of that channel will be loaded from the port memory data bus or from the XXBUS with the page address of the block to be transferred. Once loaded, the page pointer remains static throughout the DMA block transfer process of its channel.

Both DMAM block transfer channels will have dedicated transfer address pointer registers which include the Active DMA Pointer Register (ADMAP) and the Passive DMA Pointer Register (PDMAP). These two pointers will be 16-bit counters with parallel load capability. The output of these registers will be mapped into the least significant bit of the port memory address bus while

the inputs are mapped into bits 3 thru 18 of the XTB. Prior to a DMA block transfer process by a channel, the starting address of the data block to be transferred will be loaded from the Port Memory Data Bus or from the XXBUS into the channels DMA pointer register. During the DMA block transfer process, this pointer will be incremented by one while the word count register is decremented by one after each execution of the channels DMA block transfer MI directive. This operation is repeated until the channels's word counter reaches a count of zero.

Each DMAM block transfer channel will have dedicated word counters which include the Active DMA Word Counter (ADMAWC) and the Passive DMA Word Counter (PDMAWC). These word counters will be 16-bits in width and have a parallel load capability. Each word counter will also have a status line that indicates to the MCM when the contents of the word counter equals zero. The inputs of the word counters will be mapped into bits 3 thru 18 of the XTB. Prior to a DMA block transfer process, the word counter of that channel will be loaded from the port memory data bus or from the XXBUS with the length of the data block to be transferred. During the DMA block transfer process, the word counter will be decremented by one after each execution of the channels DMA block transfer MI directive until the word counter reaches the count of zero. When a word counter reaches a count of zero, its terminal count status line (ATC/PTC) will become active. ATC/PTC are used in conjunction with the JMP (ATC/PTC) MI directive to detect end of DMA transfer. Also, when ATC/PTC become active, the DMA module will not execute any subsequent DMA block transfer MI directives for that channel until the word counter is reloaded for the next DMA block transfer.

4.4.4.2 DMA Single-Word Addressing. The DMAM will contain one additional address pointer. This pointer is provided by the High Byte and Low Byte General Purpose Address Pointers (HBGPAP and LBGPAP), and the MI directive Data Handling Register (MIDHR), which supports single word DMA at any time by either channel. The HBGPAP will be an 8-bit register with its outputs mapped into bits 8 thru 15 of the port memory address bus and with its inputs mapped into bits 0 thru 7 of the XTB. When located from the Microprogram Memory (MM), this value will be stored in the data field of the load HBGPAP MI

directive. Once loaded, the value loaded remains unchanged until the next load HBGPAP MI directive is executed.

The LBGPAP will be an 8-bit gate that connects bits 0 thru 7 of the XTB to bits 0 thru 7 of the port memory address bus. Supporting the LBGPAP during single word DMAs is the MI directive data handling register, (MIDHR). The MIDHR will be an 8-bit register with its input mapped into bits 0 thru 7 of the XTB and with its outputs mapped into bits 0 thru 7 of the port Memory Address Bus. During every DMA MI directive load cycle, the data field of the MI directive section will be gated into the MIDHR. During DMA single-word transfers, it is this value that is gated onto the port memory address bus by the LBGPAP. During execution of load direct MI directives, the value loaded into the MIDHR is available for transfer to any XXBIU.

4.4.4.3 DMA Module Control. DMAM control is distributed among three separate control units. These Control units are shown in Figure IV-17 and include the Port Bus Acquisition Unit (PBAU), the DMA Module Control Unit (DMAMCU) and the MI directive Loading Unit (MILU). These units will provide the DMA module with the necessary mechanisms for coordinating its activities with the PPPU and the other elements of the XXBIU and for controlling the execution sequencing of the DMAM MI directives.

4.4.4.3.1 Port Bus Acquisition Unit. The Port Bus Acquisition Unit (PBAU) will perform the operations of port bus acquisition and port bus relinquishment. A single bidirectional handshaking line, PS $\overline{RQ}/\overline{GT}$, and a three-phase protocol, request-grant-release, will be used to coordinate these operations with the other users of the ports memory bus. The required communication between the PBAU and the DMAMCU is provided by the lines GETPB, RELPB, and PBCAVL. GETPB and RELPB will be used respectively by the DMAMCU to tell the PBAU when to get control of the Port Memory Bus (PMB) and when to release it. PBAVL will be used by the PBAU to indicate to the DMAMCU when it has control of the port memory bus.

4.4.4.3.2 DMA Module Control Unit (DMAMCU). The DMAMCU will perform operations facilitating MI directive decoding and execution sequencing.

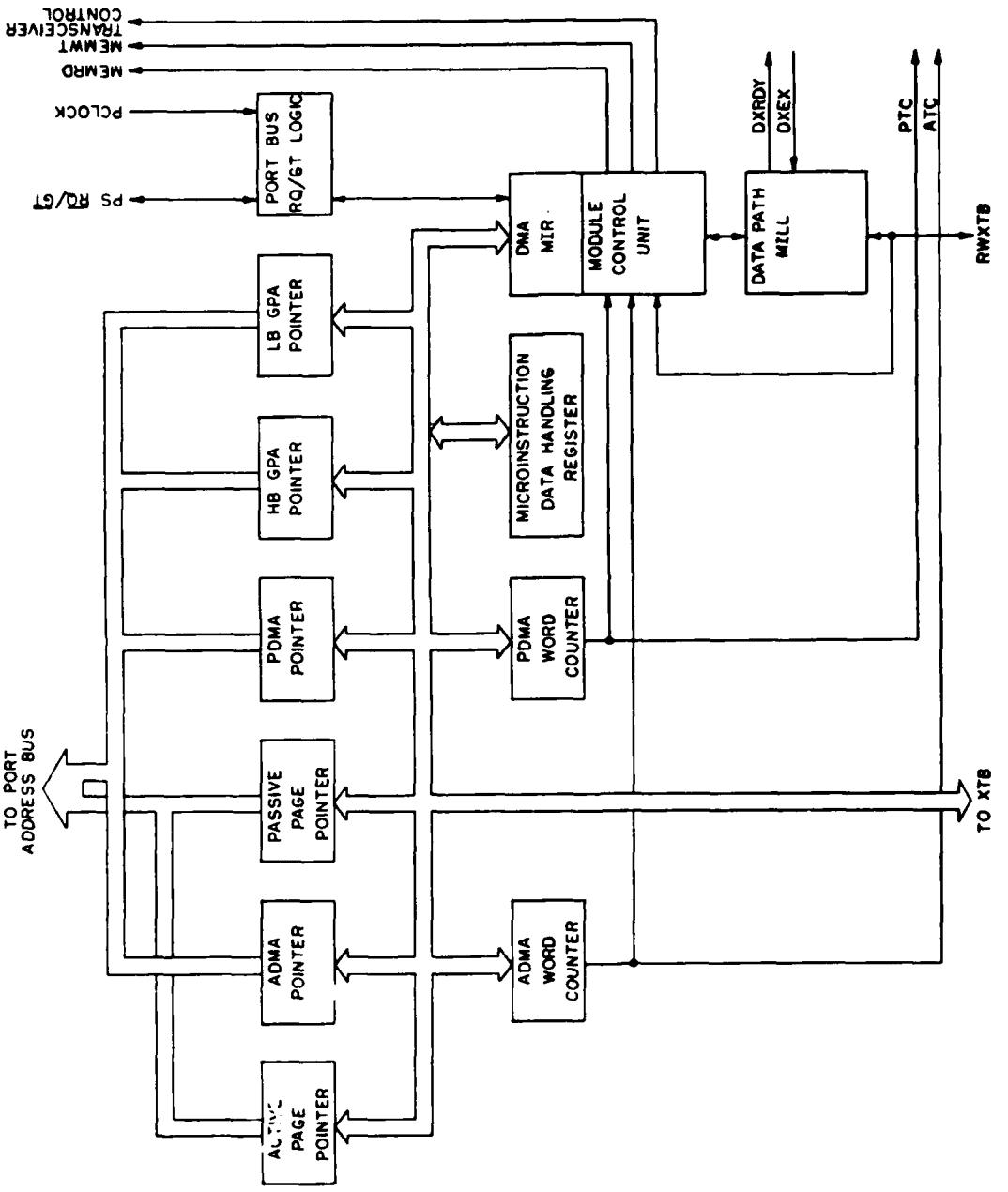


Figure IV-17. DMA Module Functional Block Diagram

Decoding is the process of determining the type of data transfer to perform. This includes determining the source, destination, and direction of involved buses. Execution sequencing will include proper read/write timing and coordinating external elements that are involved. This coordination with the port memory bus will be provided by the PBAU. With the XBM, it is supported by the bidirectional handshaking line RWXTB and with the MCM, it is supported by the handshaking lines DXEX and DXRDY.

4.4.4.3.3 MI Directive Loading Unit. The MI Directive Loading Unit (MILU) will support synchronization of the activities of the DMAM with those of the XBM and MCM during MI directive loading. The handshaking lines DXRDY and DXEX will support this synchronization between the DMAM and the MCM, and the RWXTB bidirectional line will support this synchronization between the DMAM and XMB.

4.4.5 Data Paths. This section covers the routes in which all information associated with the XXBIU is transferred from, or to, its various internal and external elements. These paths, as illustrated in Figure IV-08, include the XXBIU Transfer Bus (XTB), Port ID Data Path, Port Memory Bus, the Microprogram Memory (MM) address and data buses, and the MPC XXBUS.

4.4.5.1 XXBIU Transfer Bus. The XXBIU Transfer Bus (XTB) will be the path over which all information associated with the DMAM and the XBM is transferred. This information will include the MI directives for both the modules as well as the information that is transferred between the port memory bus and the XXBUS.

The XTB will be structured to support two types of sequences and controls. These include the MI directive execution and MI directive load.

4.4.5.1.1 Sequence. The XTB will be 19 bits in width and supports two types of bit formats, one for the MI directive execution sequence, and the other for the MI direct load sequence.

4.4.5.1.1.1 MI Directive Execution Sequence Format. In this sequence, bits 3-18 (16 bits), will contain data that can be transferred either between two modules, or between either module and port memory via the port data bus. Bits 0-2 will not be used in this sequence.

4.4.5.1.1.2 MI Directive Load Sequence Format. This format will contain bits 8-18 (11 bits) which comprise the MI directive, and bits 0-7 containing a literal value which can be transferred from the Microprogram Memory (MM) to any element associated with the XTB. During the load sequence, this value will be loaded into a holding register located in the DMA module where it is held until the execution sequence. If at that time, a load direct instruction is executed, the contents of this holding register will be transferred to the target specified in the MI directive. This requires that bits 0-2 need only be connected to the DMA module and the buffer that interfaces to the MI directive data bus.

4.4.5.1.2 Control. Both MI directive execution and MI directive load sequences will have their own respective points of control, which include the DMA module (DMAM) and the port memory bus.

4.4.5.1.2.1 MI Directive Execution Control. The point of control during this sequence will be from the DMAM which coordinates the sequence with the XBM via the low level control lines, and with the port memory bus, using the Request/Grant ($\overline{RQ}/\overline{GT}$), Memory Read (MEMRD), and Memory Write (MEMWT) lines.

4.4.5.1.2.2 MI Directive Load Control. Control during the MI directive load sequence will be distributed between all three modules which coordinate their activities via the low-level control lines that connect all three.

4.4.5.2 Microprogram Memory Address Bus. The Microprogram Memory Address Bus (MPMAB) will carry the address of the instruction to be executed from the MCM to the MM, and will consist of 11 bits thus providing an addressing capacity for 2048 MI directives. The microprogram address will contain 5 address bits, while both the MP module and the microinstruction address will contain 3 bits of address each. The detailed nature of these fields is

presented in the sections on the MM and the MCM. Control of the MPMAB will be maintained by the MCM. Since the MPMAB is dedicated, no control lines to the memory are required. Synchronization of address updates with MI directive loads will be controlled by the MCM.

4.4.5.3 Microprogram Memory Data Bus. The Microprogram Memory Data Bus (MPMDB) will carry information from the MM to the buffer interfacing to the XTB and to the MCM. It will consist of 24 bits, of which 19 bits will be bused to the buffer and carry the MI directive and data to the XBM and DMAM. Bits 0-8, 11, 19, and 21 will be connected to the MCM.

The Microprogram Memory Data Bus will be dedicated and, therefore, no control is required. The value of the data will change as the address is transferred through the memory. Loading of the microinstruction will be controlled by the MCM.

4.4.5.4 Port ID Path. The data path associated with the Port ID Header will connect the Port ID not only to the Address Recognition Unit (ARU) but also to the Port Data Bus. This buffer will be mapped into the I/O space of the port memory bus and provide the mechanism by which the PPPU can read the port's own ID. This path will be 6 bits wide and mapped into the lowest 6 bits of the Port Data Bus. The remaining 10 bits of Port Memory Bus will be forced to their zero state during a port ID read.

4.4.5.5 Port System Bus. This bus is external to the XXBIU and will contain the paths in which the Primary Port Processing Unit (PPPU), Port System Memory, and the XXBUS exchange information. The XXBIU has been designed to interface to a Port System Bus with the IOW, IOR, MEMRD, and MEMUT used as timing and control lines. Transfer of control to this bus will be accomplished using the dedicated PS $\overline{RQ}/\overline{GT}$ control line. The timing specification for these control lines will be compatible with the PPPU.

4.5 IPC Improved Firmware Design. Each MPC Port must be able to establish and participate in Port-to-Port dialogues. The hardware that provides this capability is the MPC Exchange Bus and related interfaces. The software that provides this capability is the Inter-Port Communications (IPC) Subsystem. IPC logically resides between the Bus hardware and the remaining MPC subsystems contained within each port. No other subsystem within a port directly addresses the Bus Interface Hardware. In this manner, IPC essentially envelopes the Exchange Bus interface hardware and shields the remaining subsystems from the intricacies of the Exchange Bus interface. IPC also maintains various dialogue and error counts which can be used to monitor system performance.

4.5.1 Dialogue Concepts. A copy or element of IPC resides in each MPC port. This provides a standardized communications capability to the remaining software within each local port. A subsystem within one port may express a requirement to exchange data or directives with a subsystem in another port, thus establishing a connection between the two ports, transferring the data or directives, and disconnecting the two ports. This process constitutes a port-to-port dialogue. The IPC element in the receiving port transfers the data to the software subsystem residing within that port. No intermediary port is required to establish a dialogue. Therefore, many port-to-port dialogues may occur in a given time frame.

4.5.1.1 Multiple Dialogue Requests. Several types of port-to-port communications are supported by IPC. In some cases, a period of time may elapse between the request for, and the completion of, a specified transfer. At times more than one dialogue request may be outstanding. IPC will handle several requests by multiplexing dialogues to different ports through its single physical bus interface.

A port may be involved in two types of dialogues during any given time frame. It may be engaged in a dialogue that was initiated by another MPC port (a dialogue in the passive role), while at the same time be engaged in a

port-to-port dialogue that was initiated by itself (a dialogue in the active role). While a port may participate in each role simultaneously, it may not be engaged in two active or two passive roles concurrently.

4.5.1.2 Dialogue Synchronization. Specific physical details relevant to the command structure of the bus interface hardware, timing required for interport synchronization, and the hardware attributes of each local port and microprocessor are totally resolved within IPC. The multiple processor nature of the MPC is transparent to the remaining software subsystems. IPC allows the subsystems in each MPC port to concentrate exclusively on its own logical requirements, yet be able to solicit inputs, address outputs, and initiate processing in other external MPC ports.

4.5.1.3 IPC Communication Modes. IPC allows any subsystem to exchange data and directives with another cooperating subsystem by establishing a port-to-port dialogue. Within this dialogue one port will be the requester and one port will be the responder. The requester is the port that detects the current need for a dialogue and performs the first action in an attempt to initiate the dialogue. The responder is the port that has been conditioned to participate as the other member in a particular dialogue. Normally, the responder is the receiver of the data or directives produced by the requester. The relationship between the requester and the responder ports will determine the communication mode.

4.5.1.4 Direct and Indirect Communication Modes. All port-to-port communications may be divided into two modes, depending on whether the requester or responder port controls the dialogue. The requester port may need to directly contact the responder port, establish a connection, and carry out the required dialogue. Alternatively, the requester may choose to notify the responder of the required dialogue and defer the actual contact and dialogue control to the responder port. Within IPC these two distinct situations determine the direct and indirect communication modes respectively. In the indirect mode, the requester is willing to wait for the responder to detect the notification and initiate the dialogue. In the

direct mode, the requestor initiates the dialogue immediately, waiting only if the responder is not available.

4.5.1.5 Parallel Tasking. The architecture of the MPC imposes a natural distribution of functions and allows the assignment of dedicated hardware resources to each individual function. Each port is dedicated to a single function, and the complicated multitasking of a central processor is not required. However, individual functions may still break down into multiple, parallel tasks.

4.5.1.5.1 Port Activities. Each independent parallel task will be called an activity. Each MPC port may contain any number of activities as dictated by the user level logic contained within the port. At any point in time, each individual activity may have its own communications requirements.

4.5.1.5.2 Communication Channels. Each activity requiring port-to-port communications will require a communications channel. In any dialogue, a specific communication channel in one port will be connected to a specific communication channel in another port. A communication channel is thus a long term control mechanism that will allow a specific activity to conduct a dialogue.

4.5.1.5.3 Channel Control Table (CCT). A port will typically contain several communication channels. The current status of each communication channel is described in a Channel Control Table (CCT) dedicated to that communication channel. The location of the CCT is specified by the subsystem. To carry out a dialogue, the activity enters information describing the dialogue in its CCT and then calls IPC which uses the information in the CCT to carry out the dialogue. All CCTs have the same structure, which allows IPC to interface with varying port subsystems through a standardized interface.

4.5.1.5.4 Port Control Table (PCT). Information concerning the status of the port as a whole is maintained in the Port Control Table (PCT). The PCT also contains items used strictly by IPC during a dialogue.

4.5.1.5.5 Dialogue Multiplexing. Because a port may contain several CCTs, IPC is able to multiplex dialogues for many different activities via the port's single Exchange Bus Interface Unit (XXBIU). Each activity is provided the capability to carry out port-to-port communications with no awareness of the existence or state of any channel other than its own.

4.5.1.6 Dialogue Termination Codes. At the end of every dialogue, IPC will always tell the user if the dialogue completed successfully or if the dialogue was incomplete and why it failed. This is accomplished by means of a dialogue completion code which is stored in the user CCT by IPC when a port is engaged in the passive role. When a port assumes the active role IPC will return the completion code directly to the user logic.

4.5.1.7 Dialogue Error Count. In order to facilitate system performance monitoring and hardware failure detection, IPC maintains a record of the number of incomplete dialogues which terminated due to various causes. The number of dialogues completed through a given channel is stored in that channel's CCT. The number of terminated dialogues as a whole is stored as a function of the dialogue termination code in the PCT.

4.5.1.8 IPC Dialgue Counts. IPC also maintains a record in the PCT of several anomalous situations which may be indicative of error situations. These are maintained separately because they may or may not be reflected in the counts maintained on the basis of the dialogue termination code.

4.5.1.9 Dialogue Control Words (DCW). A DCW is a block of data which contains the starting address and the length of the data to be transferred, as well as the control information for the next segment of data. The control information contains the end of dialogue indicator, the direction flow of the next data segment, and other control information necessary to maintain dialogue coordination.

4.5.1.10 Data Segments. One or more data words within a segment must be transferred across the exchange bus in the same direction. The direction of the data flow can only be reversed at the end of a segment.

4.5.2 Dialogue Contention. Resolution of port-to-port contention, which is currently managed totally in IPC, will migrate to the Exchange Bus Interface Unit (XXBIU) of each MPC port through the use of the Lockon Request Enable flag (LORQEN) located in the microprogram control module of the XXBIU. The setting of LORQEN will reflect the ports willingness to participate in a port-to-port dialogue in a passive role to the XXBIU hardware. Upon receiving a lockon request from an active port, the XXBIU will schedule the Passive Lockon Response Microprogram (MP) of IPC for execution, only if LORQEN reflects a set condition. If a lockon request is received with LORQEN in a clear condition, the XXBIU will not schedule the Passive Lockon Response Microprogram. This will force the XXBIU in the active port to invoke a lockon request time-out condition. IPC in the active port will recognize this time-out condition indicating that the target port is currently engaged in a dialogue in the passive role. At this time IPC will pass a port busy status to the requesting user activity. IPC controls the set or clear condition of LORQEN at all times, enabling each MPC port to maintain its own passive availability independently of other ports in the MPC system.

4.5.3 Active and Passive Port Roles. In any dialogue, regardless of the communications mode, one port will take the responsibility for actively contacting the other port and controlling the dialogue. The other port must passively wait to be contacted, and if the contact is accepted, must be prepared to be driven through the dialogue under the active port's control.

Therefore, in any dialogue, one port will play the active role while the other port will play the passive role. Since these roles require almost diametrically opposed behavior from the IPC elements in the connected ports, IPC is divided into two major components, active and passive. Currently, the active component of IPC is executed as a result of a call from a local port activity and performs all necessary bus operations. The passive component of IPC is executed as a result of interrupts being received from the active component in another port. Section 3.2.1.1.5 discusses the current implementation. In the improved firmware design, the active component of IPC is still initiated as a result of a call from the local port subsystem and

passive IPC will still be initiated by active IPC at the interrupt level. However, after both passive and active roles of a dialogue are initiated, both sides are interrupt driven in the form of request, and response interrupts. Active IPC will issue all request logic to a passive port at the interrupt level since it controls the port-to-port dialogue. Passive IPC will execute all response logic to the active port at the interrupt level. Therefore, after dialogue initiation by active IPC, both the active and passive ports drive each other in a synchronous manner. The interrupt structure of each port will allow both active and passive IPC to execute in the same time frame, thus allowing one port to be engaged in two dialogues simultaneously (one active, and one passive).

4.5.4 Logical Levels of IPC. Both the active and passive components of IPC are divided into two distinct levels of operation which will be referred to as IPC Level One (IPC L1), and IPC Level Two (IPC L2). The execution of each level will run asynchronously to the other. IPC L1 will be responsible for the control and execution of all XXBUS data transfer operations, and is the lower of the two IPC levels. IPC L2 will initate the various IPC L1 functions in a structured manner, while also interfacing user activity processing routines during each dialogue transaction as required (Figure IV-18).

4.5.4.1 IPC Level One Description. IPC L1 will consist of microinstruction code residing in Microprogram Memory of the Exchange Bus Interface Unit (XXBIU), in each MPC port. IPC L1 is divided into three logical functions, consisting of port-to-port lockon, data transfer, and dialogue termination. Each function is further broken down into various microprograms (MP), with each MP dedicated to an active or passive role within a function. Each IPC L1 function will be initiated by IPC L2 processing routines through the use of port processor I/O commands. Each MP within an IPC L1 function will initiate MPs in another port that reside within the same function. MPs in an active port will initiate MPs in the passive port, and vice-versa, thus establishing a synchronous request/response method used between any two ports engaged in a dialogue. MPs within each function may also initate IPC L2 processing routines, through the use of interrupts to the Primary Port Processing Unit (PPPU) when additinal processing is required to continue the

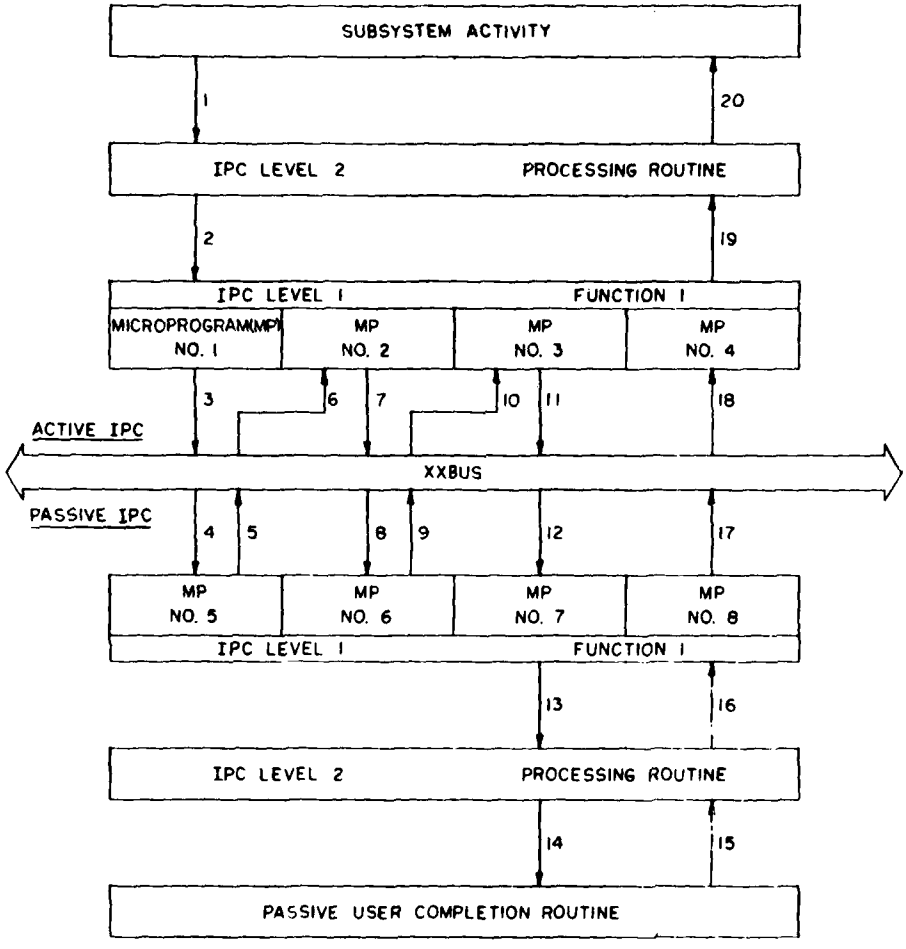


Figure IV-18. IPC Logical Levels

function, or the function is complete. IPC L2 will maintain continuity with IPC L1 at the point of each interrupt via the use of status registers, located in the XXBIU, which will be updated by the IPC L1 function prior to issuing the interrupt to IPC L2, thus transfering control and status information needed by each IPC L2 interrupt routine for proper execution.

4.5.4.2 IPC Level 2 Description. IPC Level 2 (IPC L2) resides in the read only memory (ROM) of each MPC port and is executed by the port processor. All bus transactions are handled by initiating IPC L2 functions via port processor I/O commands to the XXBIU in which the IPC L1 microcode resides. IPC L2 will return all port dialogue statuses to their respective cells in the PCT and CCT. This level will also pass intermediate statuses as required during a dialogue to facilitate necessary port processing.

4.5.5 IPC Level 1 Program Description. IPC Level 1 (IPC L1) will be divided into three dialogue functions: lockon, segment transfer, and termination. The lockon function will connect and synchronize the two ports for a port-to-port dialogue. The segment transfer function will move data or directives between the two ports engaged in a port-to-port dialogue. Termination will return both ports to their respective subsystems.

4.5.5.1 Lockon Function. This function will be initiated by the IPC L2 lockon routine and is divided into six microprograms (MP). Each MP will be dedicated to an active or passive role. The two ports to be engaged in the dialogue will be synchronized and the communication channel to be used is verified. Upon completion of this function the two ports are ready to begin the segment transfer function (Figure IV-19).

4.5.5.1.1 Active Lockon Request MP. The target port's bus address and associated routing data will be moved from the PCT to the Exchange Bus Interface Unit (XXBIU) to set up for future bus write commands. The bus address of this port as well as the bus routing data to be used by the passive port will then be written across the XXBUS to the Lockon Response MP residing in the destination (passive) port. The Lockon time out timer will

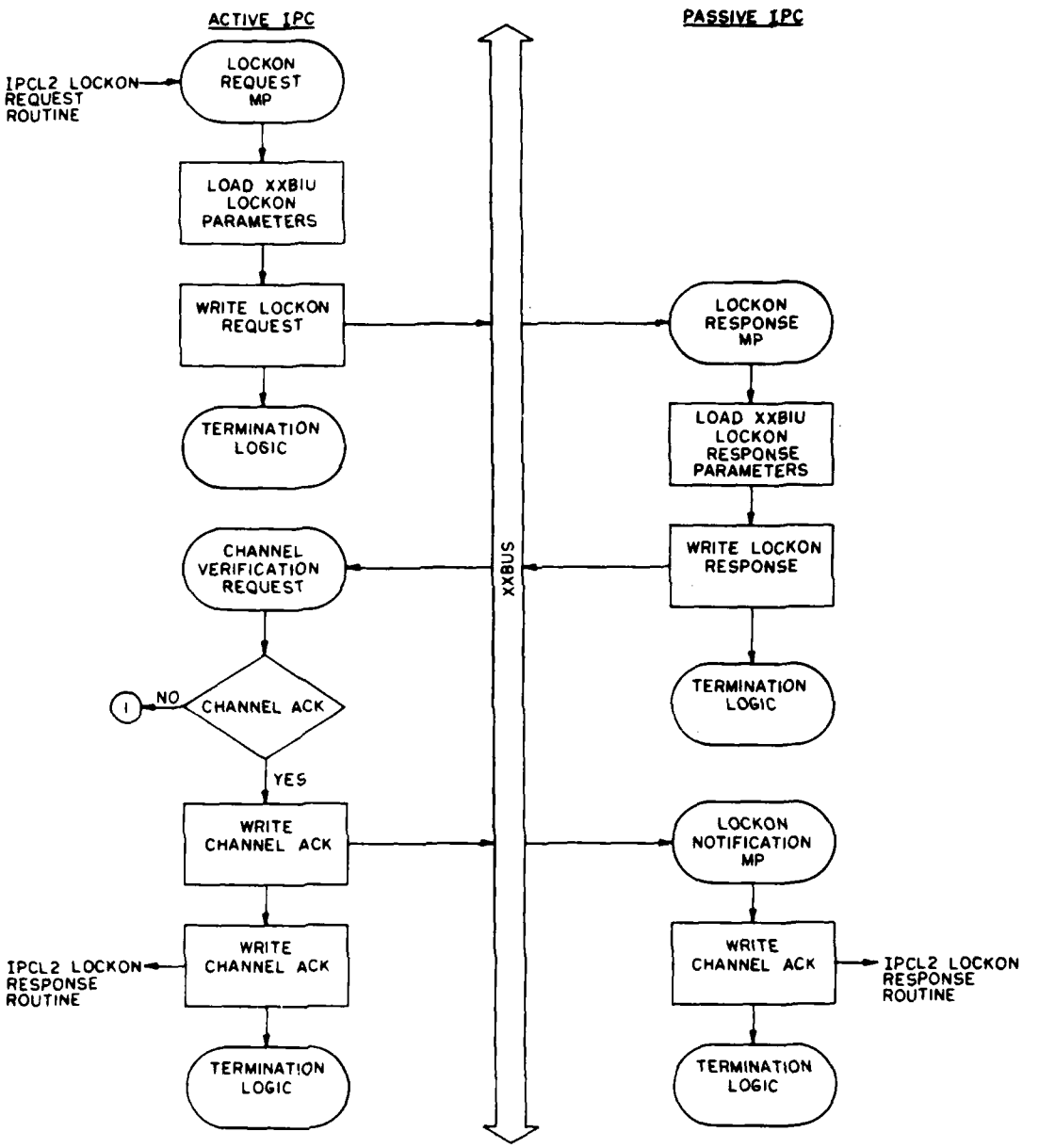


Figure IV-19. IPC Level One Lockon Function (1 of 2)

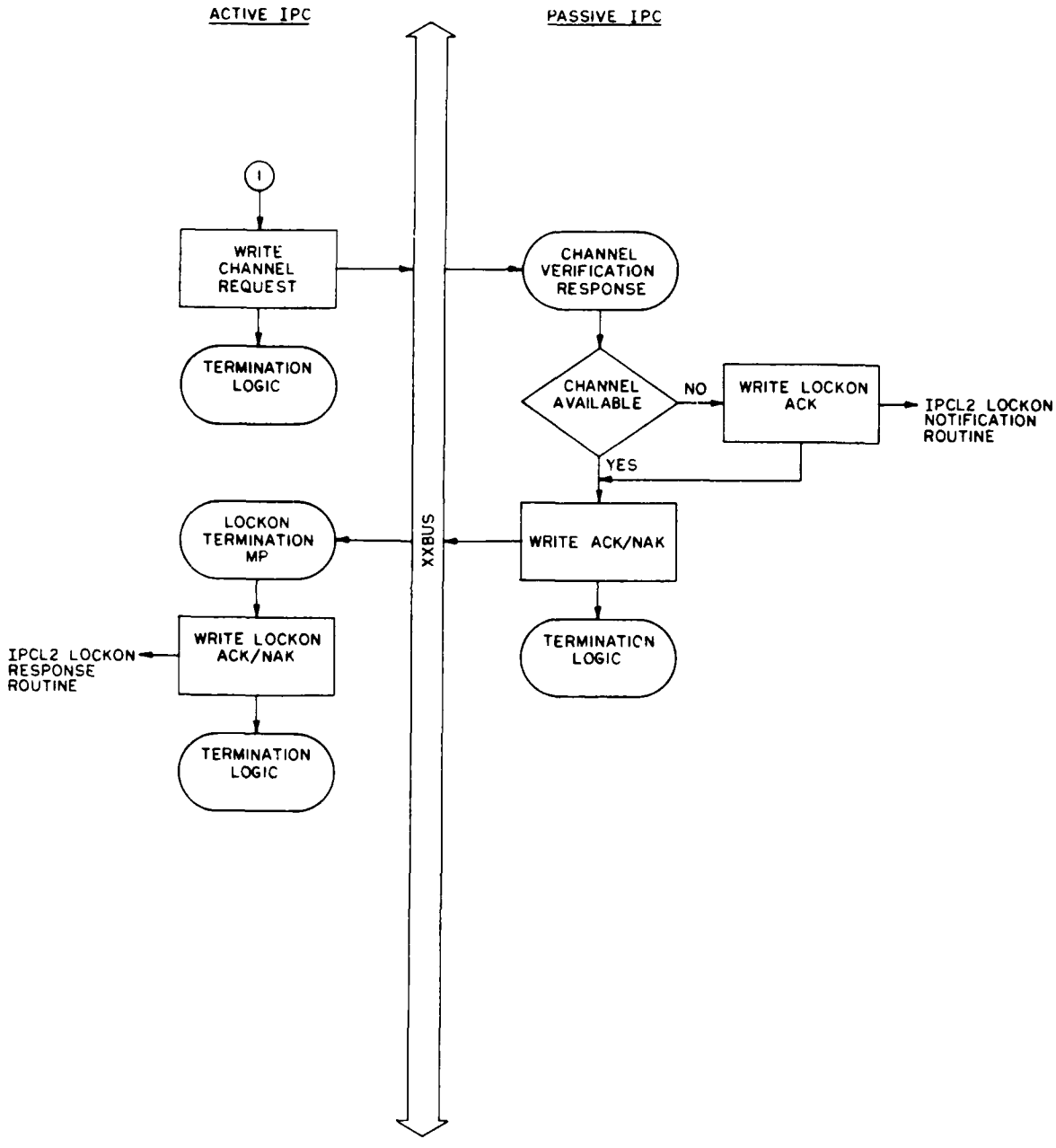


Figure IV-19. IPC Level One Lockon Function (2 of 2)

then be started, and this MP is terminated, waiting for the destination port to reply with a passive lockon response.

If the passive lockon response is not received within the time normally required to receive a response, the XXBIU will issue an IPC L2 interrupt to the PPPU. This condition indicates that the requested port is currently busy in a dialogue in the passive role. If a lockon response is received, the time-out timer will be reset, thus never allowing the time-out interrupt to occur.

4.5.5.1.2 Lockon Response MP (Passive). The fact that this MP is executed indicates that this port is available to participate in a port-to-port dialogue in the passive role. The Port Bus address and associated bus routing data of the active port that issued the lockon request will be moved first from the input data section of the XXBIU to the XXBIU hardware associated with the port destination of future XXBUS write commands to be issued. The first available passive communication channel will be moved from the PCT to the XXBIU, and written via a bus write to the active ports Channel Request-Verification MP. This formulates the Passive lockon response to the active port. The port's passive availability is then lowered, indicating to the XXBIU to ignore any subsequent lockon requests while the current dialogue is in progress. The passive time-out timer is then started, and the MP is terminated.

4.5.5.1.3 Channel Request Verification MP (Active). This MP will be initiated by the lockon response MP in the passive port. The lockon response data located in the XXBIU input data section is the first of 'N' communication channels available in that particular passive port. The next phase in the lockon function will be to verify that the specific communication channel needed to conduct the dialogue matches one of the passive channels available. The communication channel required to conduct the port-to-port dialogue will be moved to the XXBIU from the PCT and compared to the first available channel in the passive port. If the two channels match, the verification process is complete. If not, a channel verification request must be made to the passive port.

If the verification process is complete, the verified communication channel will be written across the XXBUS to the Lockon Notification MP in the passive port, thus notifying the passive port of the successful port-to-port lockon process and also of the communication channel to be used during the rest of the dialogue. Section 4.5.5.1.5 discusses this microprogram. An interrupt will then be issued to the PPPU to notify active IPC L2 that the lockon function has completed successfully. The active time-out timer is started and the MP will be terminated, thus completing the lockon function in the active port.

The Communications Channel Lockon Request-Verification MP will be executed if the first available passive channel information sent as the passive lockon response was not the passive channel needed to conduct the dialogue. The required Communication channel will then be written to the Communication Channel Verification MP in the passive port. This MP will scan the rest of the available passive channels for a match and write an ACK/NAK back to the active port as a response. The active time-out timer will be started, and the Channel Request Verification MP is terminated, waiting for the passive ports verification response.

4.5.5.1.4 Channel Verification Response MP (Passive). This MP will move the remainder of the available passive communication channels from the PCT one at a time to the XXBIU and look for a match with the requested channel now residing in the XXBIU data input section, placed there during the bus write that executed this MP.

If a match is found, a Lockon ACK will be written to the Lockon Termination MP in the active port. The communication channel to be used will be stored in the PCT, and an interrupt issued to the PPPU notifing Passive IPC L2 of the successful lockon and the communication channel to be used for the duration of the port-to-port dialogue. The passive time-out timer will be started, and the MP terminated, indicating the successful completion of the passive portion of the lockon function.

If a match is not found, a NAK will be written to the Lockon Termination MP in the active port. The ports passive availability will then be raised, notifing the XXBIU that the port is now available for lockon requests written to this port, and the MP is terminated. The PPPU will never be interrupted when a lockon attempt is unsuccessful except during a time-out condition.

4.5.5.1.5 Lockon Notification MP (Passive). This MP will be executed if the first available passive communication channel sent to the active port as the passive lockon response, is verified as the channel to be used to conduct the port-to-port dialogue. This channel information, which was transferred in the bus write that executed this MP, will be moved from the data input section of the XXBIU to the PCT. An interrupt will then be issued to the PPPU to notify passive IPC L2 of the successful lockon, and the communications channel to be used to conduct the port-to-port dialogue. The MP will then terminate after the passive time-out clock is started, successfully completing the port-to-port lockon function.

4.5.5.1.6 Lockon Termination MP (Active). This MP will be executed by the Passive channel verification MP. The response residing in the input data section of the XXBIU will be an ACK or NAK value, depending upon its channel verification result. An interrupt will be issued to the PPPU to notify active IPC L2 of the ACK/NAK condition. If the response was an NAK, the MP will be immediately terminated. If the response was an ACK, the active time-out clock is started and the MP will be terminated, waiting for the first segment transfer function to be executed from active IPC L2.

4.5.5.2 Segment (Data) Transfer Function. This function will be initiated by the IPC L2 segment transfer routine. It includes the actual port-to-port data transfer from either the active to passive, or passive to active ports (active write or passive write respectively). After the data is transferred, redundant checkword values will be verified from both the active and passive ports. Passive IPC L1 will then notify IPC L2 of the successful data transfer. At this point, IPC L2 will provide any necessary port processing to be accomplished and a completion status value then will be passed to the active port, thus terminating the function. This function may be executed

sequentially as many times as needed to complete the port-to-port dialogue. Control for each segment transfer will reside in the PCT of both the active and passive ports in the form of DCWs, which are used to set up the XXBIU DMA modules of both ports.

4.5.5.2.1 DMA Parameter Load MP (Active). This MP will be initiated by active IPC L2 at the start of the segment function. After the active time-out timer is reset, the control word for the segment of data to be transferred will be moved from the PCT to the XXBIU, and transferred to the DMA Parameter Load MP in the passive port. The remaining parameters of the segment DCW, both active and passive will be moved from the PCT to the DMA module of the XXBIU in the respective ports. Both the active and passive ports are now ready for the segment DMA transfer.

The control word will then be examined to see if the data will be transferred from the active to the passive port (active write), or from the passive port to active port (passive write).

4.5.5.2.1.1 Active to Passive DMA (Active Write). Data will be moved from port memory in the active port to the XXBUS, using the DMA module of the XXBIU, and transferred to the DMA Read MP in the passive port, which in turn will transfer the data from the XXBUS to port memory. This process will be repeated until the XXBIU indicates to IPC L1 that the number of words to be transferred has been reached (Figure IV-20).

At this point, the redundant check word that was generated in the active port during the DMA transfer will be written to the Data Verification MP in the passive port. The active time-out timer will be started, and the MP terminated, waiting for a verification response from the passive port.

4.5.5.2.1.2 Data Verification MP (Passive). During the DMA transfer, the passive DMA Read MP will also generate a redundant check word. The active and passive values will be compared for verification. If the two check words do not match, a segment check word NAK will be transferred to the Segment Termination MP in the active port. The passive time-out timer will be

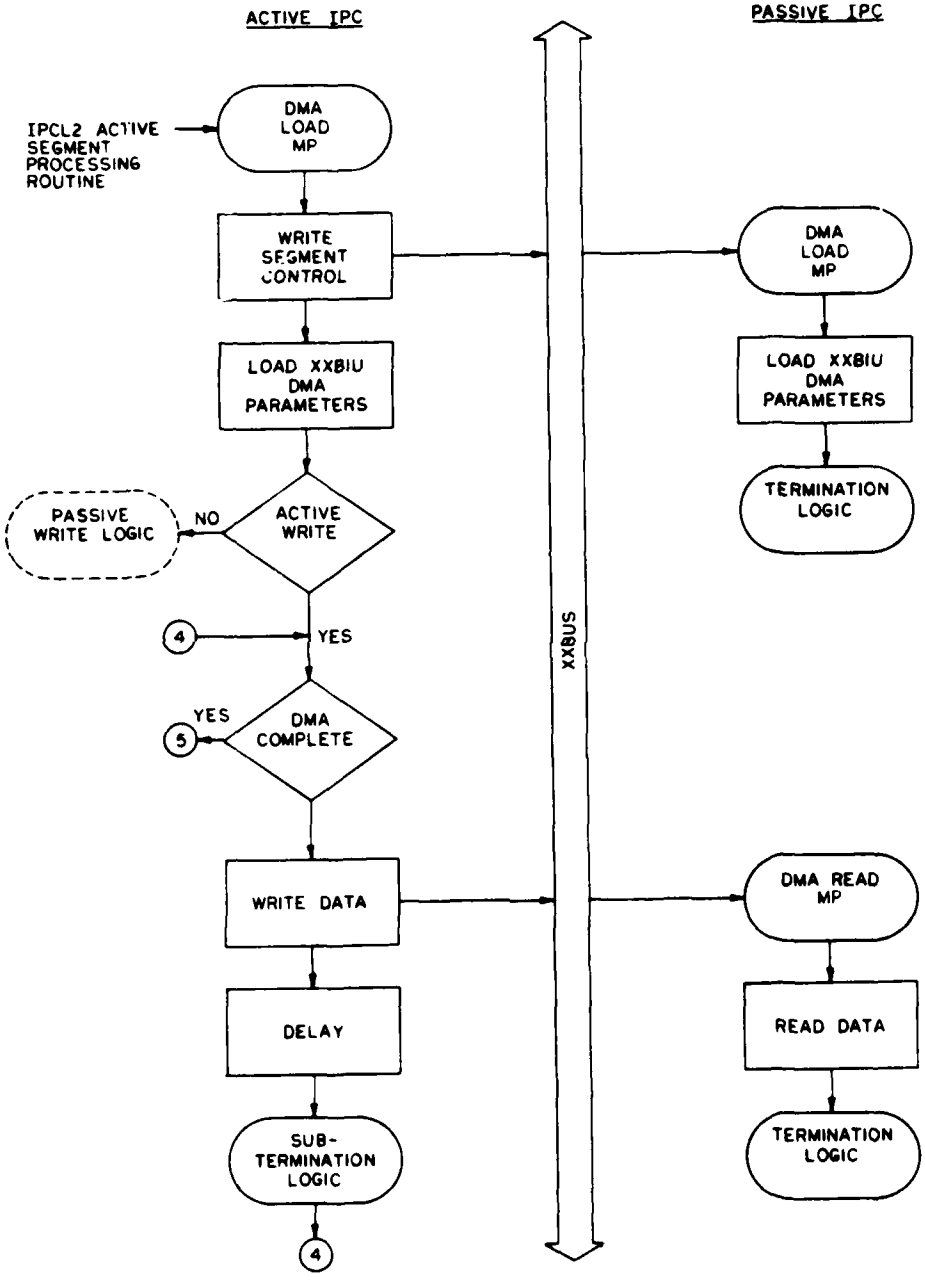


Figure IV-20. IPC Level One Transfer Function, Active Write (1 of 2)

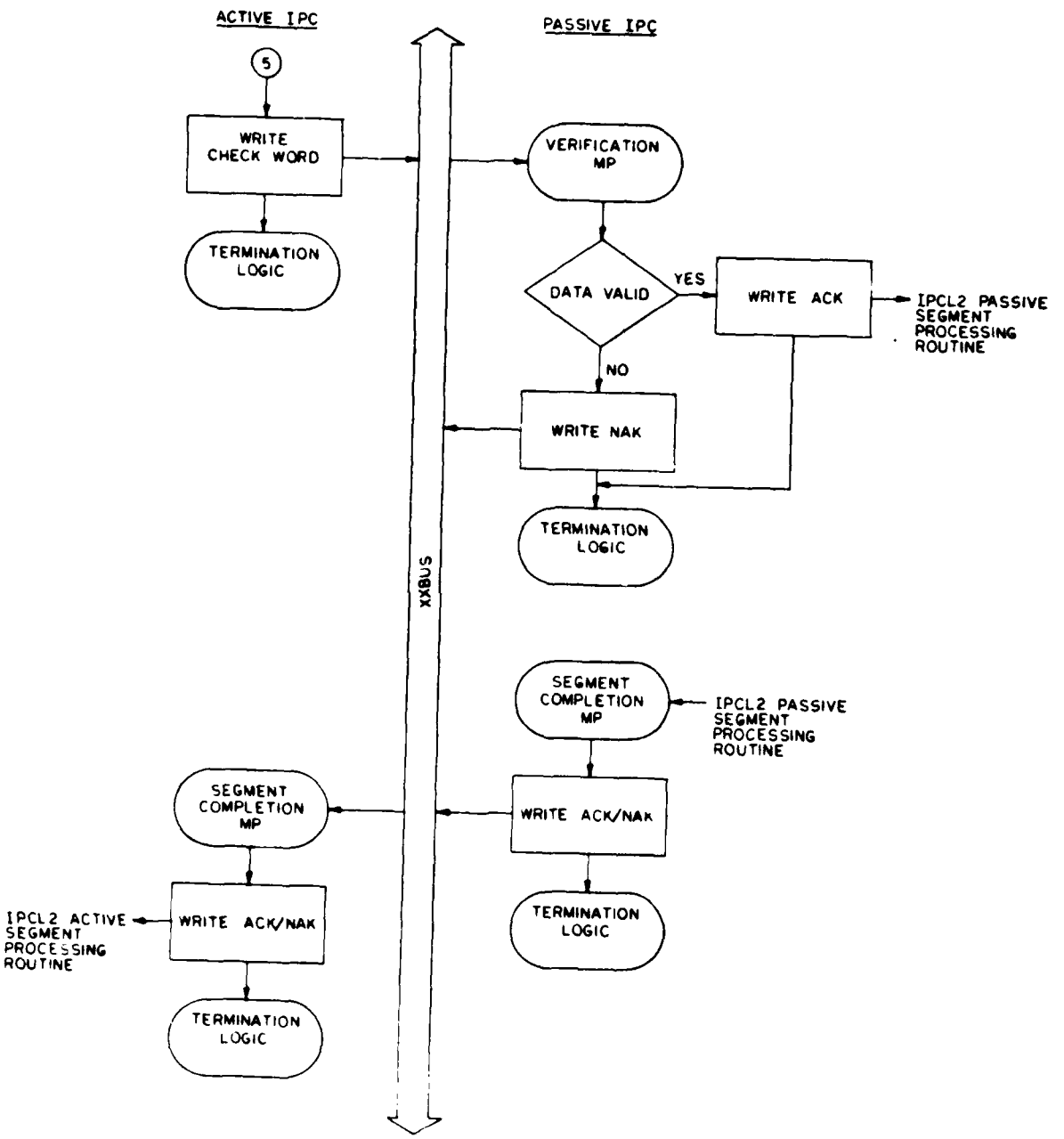


Figure IV-20. IPC Level One Transfer Function, Active Write (2 of 2)

started, and the MP terminated. Port processing will not be interrupted when a redundant check work error occurs. If the redundant check words verify, an interrupt will be issued for the PPPU to inform passive IPC L2 of the successful segment (data) transfer. After the passive time-out timer is started, the MP will terminate.

4.5.5.2.1.3 Segment Completion MPs (Passive). When all required port processing has been completed, IPC L2 will initiate one of the passive Segment Completion MPs, depending on the successful or unsuccessful segment status to be passed to the Segment Completion MP in the active port. The passive time-out timer will be started, and the module terminated.

4.5.5.2.1.4 Segment Completion MP (Active). This MP will issue an interrupt to the PPPU to notify active IPC L2 of the segment completion status transferred from the active port. The time-out timer will then be started, and the MP, as well as the segment transfer function, will be terminated for an active write.

4.5.5.2.2 Passive to Active DMA Transfer (Passive Write). After it has been determined that the data is to be transferred from the passive to the active port, the length of the segment will be written to the DMA Write MP in the passive port. The active time-out timer will be started, and the MP terminated.

4.5.5.2.2.1 DMA Write MP (Passive). Data will be moved from port memory in the passive port to the XXBUS, using the DMA module of the XXBIU, and transferred to the DMA Read MP in the active port, which in turn transfers the data from the XXBUS to port memory. This process is then repeated till the XXBIU indicates to IPC L1 that the number of words to be transferred has been reached (Figure IV-21).

At this point, the redundant check word that was generated in the passive port during the DMA transfer will be written to the Data Verification MP in the active port. The passive time-out timer will be started, and the MP terminated, waiting for a verification response from the active port.

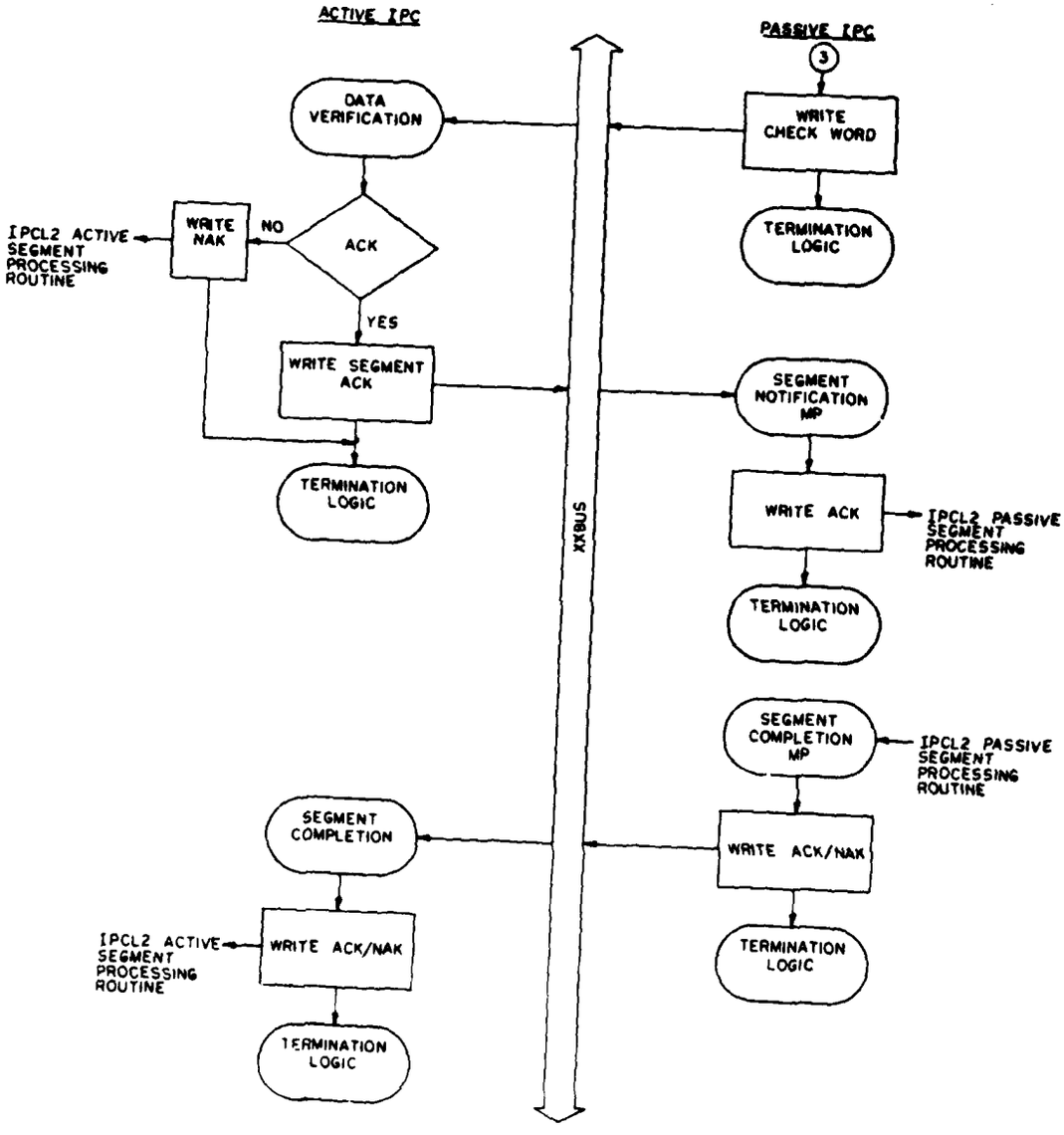


Figure IV-21. IPC Level Two Segment Transfer Function, Passive Write (1 of 2)

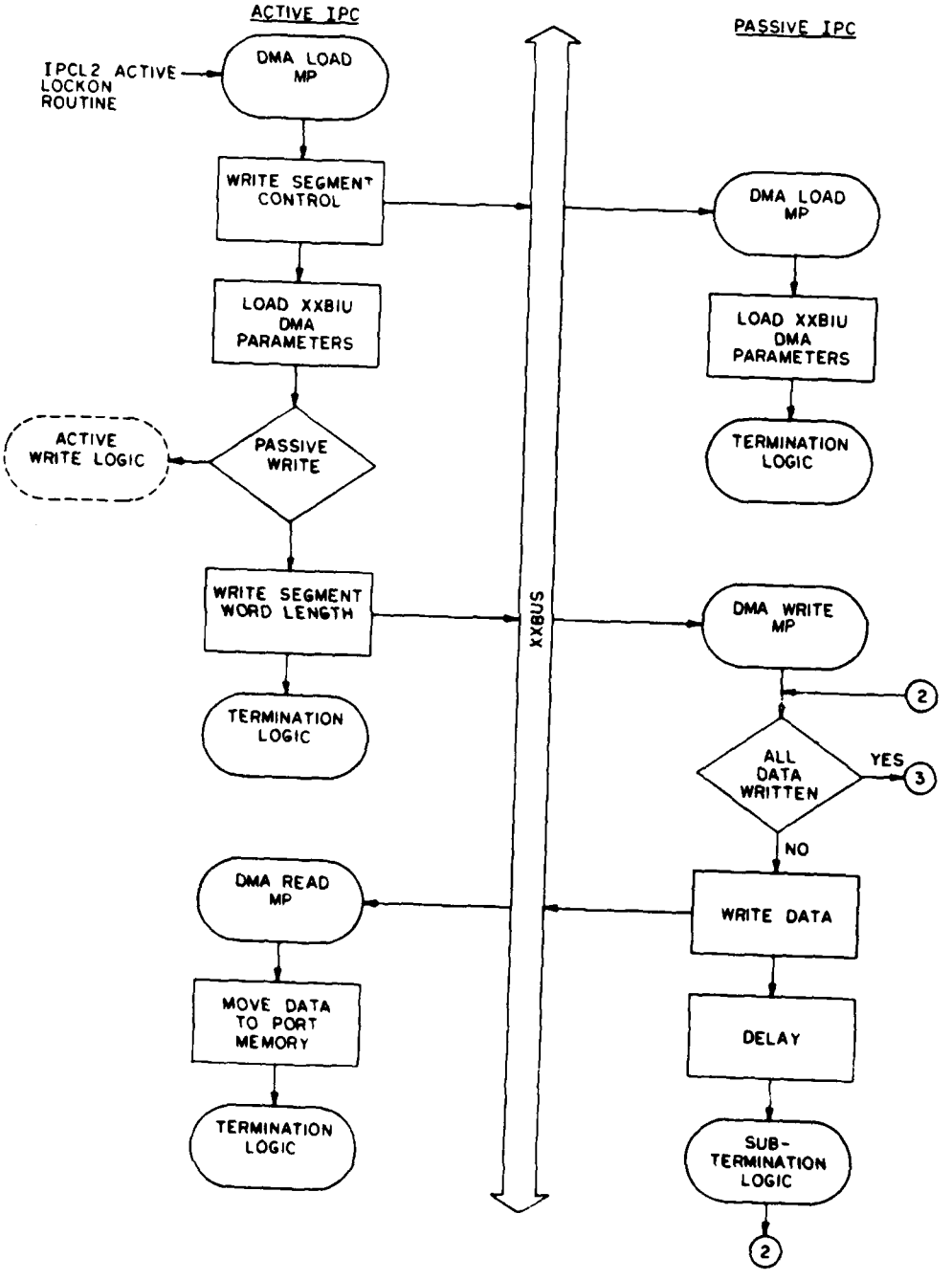


Figure IV-21. IPC Level Two Segment Transfer Function, Passive Write (2 of 2)

4.5.5.2.2.2 Data Verification MP (Active). During the DMA transfer, the active DMA Read MP will also generate a redundant check word, where the active and passive values will be compared for verification. If the redundant check words match, a successful segment transfer status will be written to the Segment Notification MP in the passive port, before the active time-out timer is started, and the MP terminated.

If the redundant check words do not match, an interrupt will be issued to the PPPU to notify active IPC L2 of the redundant check error. The active time-out timer will be started and this MP terminated.

4.5.5.2.2.3 Segment Notification MP (Passive). At this point, the active port will have verified the transferred data. An interrupt will be issued to the PPPU to notify IPC L2 of the successful segment completion. After the passive time-out timer is started, the MP terminates.

4.5.5.2.2.4 Segment Completion MP (Passive). When all required port processing has been completed, IPC L2 will initiate one of the passive Segment Completion MPs, depending on the successful, or unsuccessful segment status to be passed to the Segment Completion MP in the active port. The passive time-out timer will be started, and the MP terminated.

4.5.5.2.2.5 Segment Completion MP (Active). This MP will issue an interrupt to the PPPU to notify active IPC L2 of the segment completion status transferred from the passive port. The time-out timer will then be started, and the MP, as well as the segment (data) transfer function is terminated for a passive write.

4.5.5.3 IPC Level 1 Lockon Termination Function. This function will send a dialogue termination status (ACK or NAK) to the passive port to notify passive IPC of dialogue termination. The passive port will then echo the completion status back to the active port to verify that the two ports are still in sync, and notify the passive IPC L2 termination routine to execute its termination logic. The dialogue completion status to be transferred to the passive port will be located in the active ports PCT (Figure IV-22).

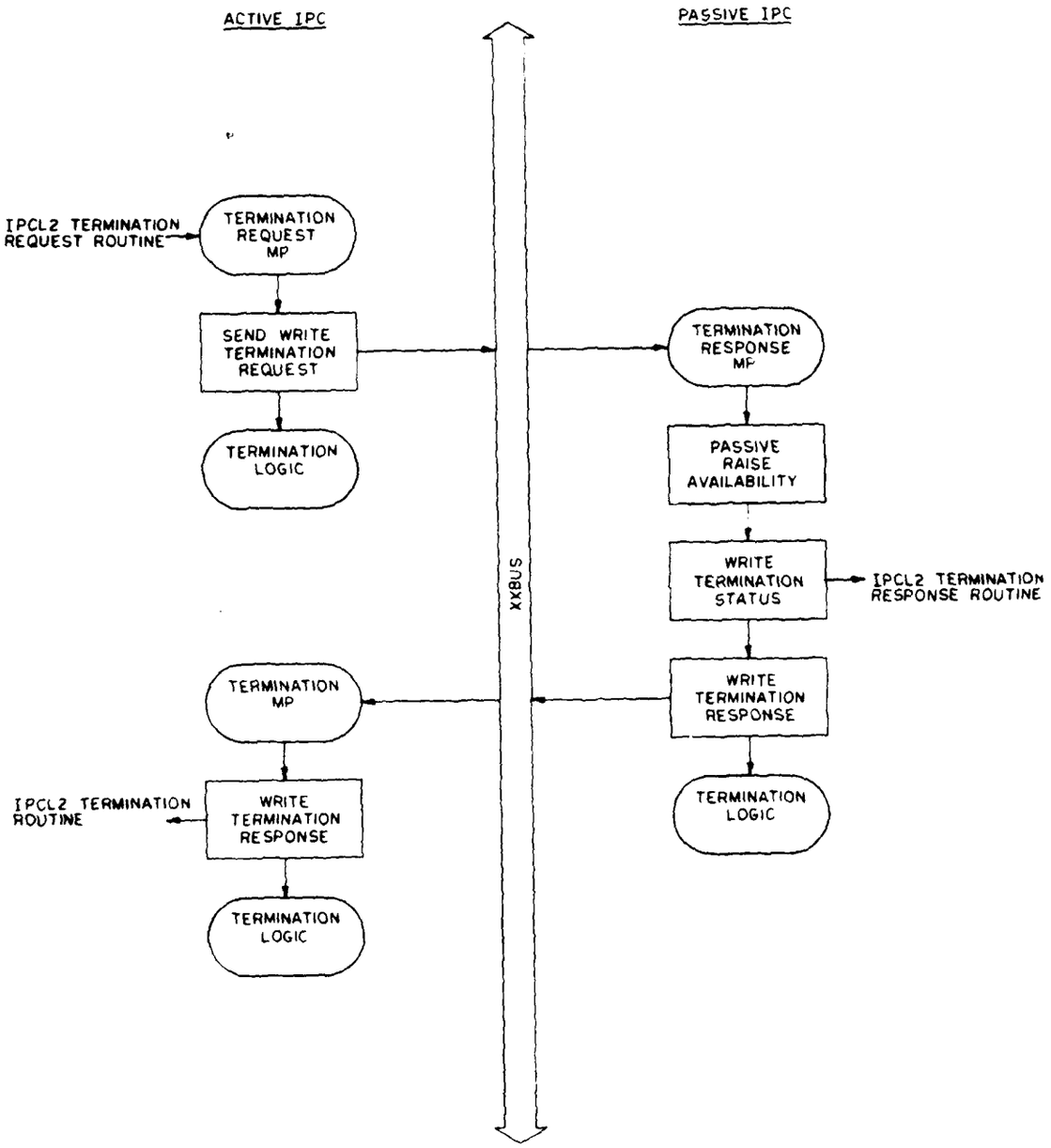


Figure IV-22. IPC Level One Termination Function

4.5.5.3.1 Termination Request MP (Active). This MP will be executed by an IPC L2 termination request routine. The dialogue completion status will be moved from the PCT to the XXBIU, and written via a bus write to the termination response MP in the passive port. The active time-out timer will then be started, and this MP terminated.

4.5.5.3.2 Termination Response MP (Passive). This MP, initiated by the Termination Request MP in the active port, will first move the dialogue completion status to the PCT. The passive interrupt will then be issued to the PPPU to notify the IPC L2 termination response routine of the final dialogue completion status. This same completion status will then be written to the Termination MP in the active port as the termination response. The passive availability flag will then be raised to allow this port to participate in future dialogues in the passive role, and the MP terminated, indicating the completion of the port-to-port dialogue in the passive port.

4.5.5.3.3 Termination MP (Active). The execution of this MP will indicate that the passive port has been notified of the dialogue termination, and the status of its completion. An active interrupt will be issued to the PPPU to notify the IPC L2 termination routine of the successful dialogue termination. The MP will then be terminated, indicating the completion of IPC L1 in the port-to-port dialogue.

4.5.6 IPC Level 2 Description. IPC Level 2 (IPC L2) resides in port memory and is executed synchronously with the user activity and may run synchronously or asynchronously with IPC L1. IPC L2 is divided into three logical sections: lockon, segment transfer, and termination. Each section may be divided into one or more routines. Each routine will be dedicated to either an active or a passive role. Any fatal error detected within the dialogue will cause the termination of the dialogue while returning the error status to the activity that requested the port-to-port transaction.

At the termination of any IPC L2 routine, either active or passive, processing control will be returned to the subsystem residing in its own

respective port, and will remain there until the next IPC L2 routine is initiated via an IPC L1 issued interrupt.

4.5.6.1 IPC Level 2 Lockon Section. The Lockon section will execute the IPC L1 lockon function and process the status information returned upon completion of that function. Any error will be returned to the subsystem that initiated the dialogue request. Upon a successful lockon by the IPC L1 lockon function the DCWs for both the active and passive ports are processed for the first segment transfer and the segment transfer function of IPC L1 is executed. The Port interrupt controller is then revectored to allow the IPC L2 segment transfer section to control further dialogue processing upon the completion of the IPC L1 segment (data) transfer function (Figure IV-23).

4.5.6.1.1 Lockon Routine (Active). The lockon routine will be initiated by the user activity via a call to IPC L2 in which the communications channel to be used to conduct the dialogue is transferred as a call parameter. The control information will be transferred fron the CCT to the PCT. This information includes the target port's bus address, the originating port's address, and the communication channel to be used by the passive port for dialogue control once port-to-port lockon has been accomplished. The bus routing data will be calculated and merged with both port's addresses. The IPC L1 lockon function will then be executed via a port I/O command, initiating the lockon request to the target port. The interrupt controller will then be revectored to ensure execution of the IPC L2 lockon verification routine at the completion of the IPC L1 lockon function.

Processing control will then be relinquished to the user subsystem until an active interrupt is issued to the PPPU by IPC L1, indicating the completion of the IPC L1 lockon function.

4.5.6.1.2 IPC L2 Lockon Verification Routine (Active). This routine will be initiated via an active interrupt issued by the IPC L1 lockon function, indicating the port-to-port lockon process is complete. The lockon status will be read from the active status register in the XXBIU. This status will

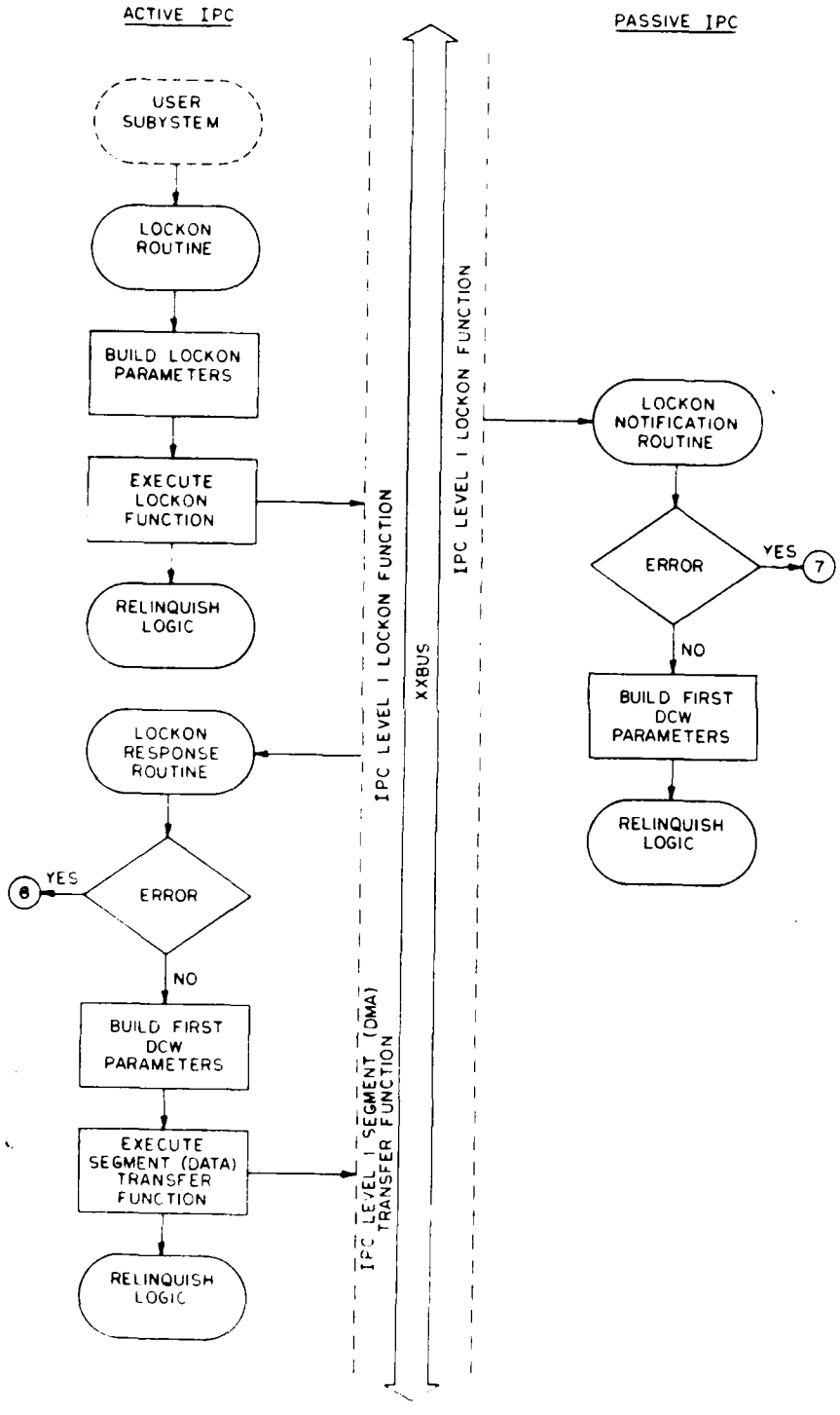


Figure IV-23. IPC Level Two Lockon Section

indicate one of four conditions: port busy, out of sync, communication channel not available, or successful lockon completion. The port busy status indicates the target port is currently engaged in the passive role with another port-to-port dialogue. Communication channel not available means that contact was made with the target port, but the communication channel required to conduct the port-to-port dialogue was not available in the target port. The out of sync status reflects the condition that contact was made with the target port, but a response was not received during the communication channel verification process of the IPC L1 lockon function. If one of these errors exist, the error status will be returned to the user activity that initiated the dialogue request, and IPC L2 terminated.

If the status reflects a successful completion, the DCW for the first segment of data to be transferred is moved from the CCT to the PCT. The IPC L1 segmemt transfer function will then be executed. The active interrupt controller will then be revectored to the IPC L2 segment completion routine, which will be initiated upon IPC L1 segment transfer function completion. This routine will then terminate, and port processing control relinquished to the port subsystem.

4.5.6.1.3 IPC L2 Lockon Notification Routine (Passive). This routine will be initiated via a passive interrupt issued to the PPPU from the IPC L1 lockon function. The lockon status will be read from the passive status register in the XXBIU, which will reflect one of two possible conditions - port out of sync, or successful lockon from an active port. The port out of sync status indicates that the passive portion of the IPC L1 lockon function never received an expected reply from the active port during the IPC L1 lockon process. This condition will result in the port out of sync error count being incremented in the PCT, the execution of the IPC L1 passive cleanup function, and termination of this routine. This will return processing control to the port subsytem at the point that the interrupt was issued.

If the status reflects a successful lockon from an active port, the passive communication channel to be used which was stored in the PCT by the

IPC L1 lockon function will be retrieved, and the DCW to be used for the first segment (data) transfer function of IPC L1 will be moved from the CCT to the PCT. The passive port is now ready to participate in the first segment transfer. The passive interrupt controller will then be revectored to execute the passive IPC L2 segment completion routine, which will be executed at the end of the IPC L1 segment transfer function. This routine terminates, and port processing will return to the user subsystem at the point the interrupt that initiated the routine was issued.

4.5.6.2 IPC Level 2 Segment Control Section. This section will control the actual port-to-port transfer of data or directives and is further divided into two routines: Passive Segment Control Routine, and Active Segment Control Routine. Any fatal error detected during this section will result in its subsequent termination and the error will be passed back to the user subsystem that initiated the dialogue (Figure IV-24).

4.5.6.2.1 Passive Segment Transfer Routine. This routine will be initiated by a passive interrupt issued by either the segment transfer function, or the termination response function of IPC L1. If the passive status register in the XXBIU reflects a dialogue termination status, processing control will be transferred to the passive termination response routine for dialogue termination processing. If the passive status register does not reflect dialogue termination the interrupt was issued by the segment function in IPC L1 indicating a segment transfer has been successfully completed. The DCW for the segment transfer just completed will be examined to see if a user completion routine is required for further processing. If the requirement exists, a completion routine will be executed. This routine will return with the segment status, which will be either a good completion status, fatal, or a non-fatal segment error status. This status is placed in the PCT and the IPC L1 segment completion function is then executed via an I/O command. If the user completion routine returned a request to terminate the dialogue, the interrupt controller will be revectored to the passive termination response routine in anticipation of a dialogue termination request. The routine will be terminated, and processing control returns to the user subsystems.

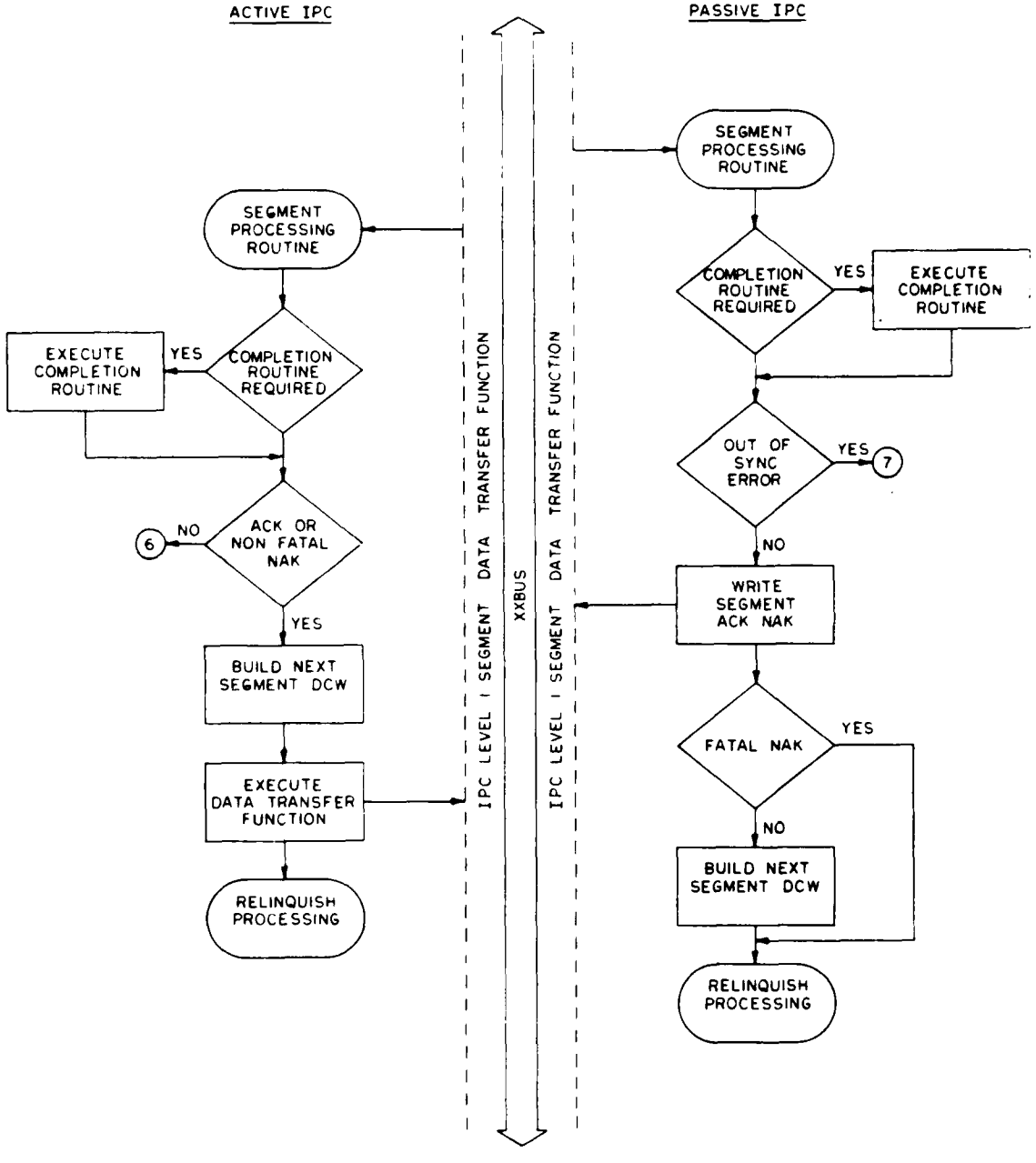


Figure IV-24. IPC Level Two Segment Completion Routine

If the segment status reflects a good completion, or a non-fatal error, the DCWs for the next segment of data to be transferred will be moved to the PCT in anticipation of the active IPC L2 segment transfer routine to execute the IPC L1 segment transfer function. This routine will then be terminated and processing control returned to the user subsystem until this routine is executed again at the end of the next segment transfer.

4.5.6.2.2 Active Segment Transfer Routine. This routine will be initiated via an active interrupt issued by the segment transfer function. The segment completion status, stored in the PCT by the segment transfer function will reflect the segment completion status. If this status reflects a fatal error, processing control will be transferred to the IPC L2 active termination request routine. If the completion status is good or nonfatal, an active user completion routine will be executed if one was requested. If it was requested, the completion status will then be rechecked to see if the completion routine opted to terminate the dialogue. If the completion status was changed to a fatal error, processing control will be transferred to the active IPC L2 termination request routine. If the status indicates a good segment completion, or a nonfatal error status, the DCWs for the next segment of data to be transferred will be moved to the PCT. The IPC L1 segment transfer function will then be initiated. This routine will then be terminated, and processing control relinquished to the user subsytem until this routine is again executed at the end of the segment transfer by IPC L1.

4.5.6.3 IPC Level 2 Termination Section. This section will terminate the port-to-port dialogue, and increment the appropriate dialogue counts. The completion status of the dialogue will be passed back to the subsystem that requested the dialogue, thus completing the port-to-port transfer process (Figure IV-25).

4.5.6.3.1 Termination Response Routine (Active). This routine will be jumped to from the segment processing routine when all segments have been successfully transferred, or a fatal dialogue error has been detected. If a port out of sync error has been detected, the error increment portion of the termination routine will be jumped to. If not, the termination function of

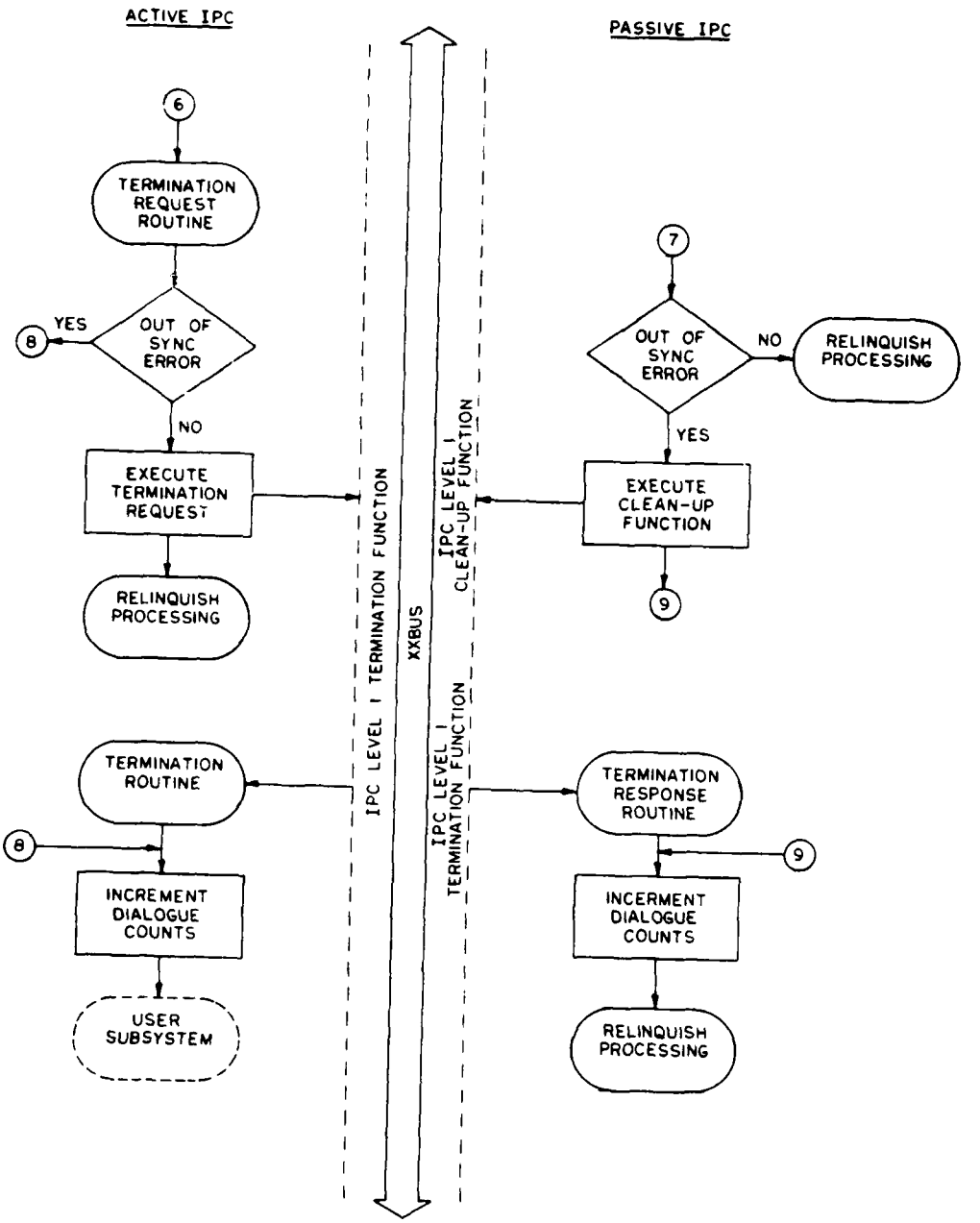


Figure IV-25. IPC Level Two Termination Routine

IPC L2 will be initiated. The interrupt controller will then be revectored to execute the termination routine upon the termination response from the passive port, and the processing control will be transferred to the port subsystem.

4.5.6.3.2 Termination Response Routine (Passive). This routine will be initiated via a passive interrupt by the IPC L1 termination function, indicating the completion of a dialogue. The completion status of the dialogue will then be incremented in the PCT, and the interrupt controller revectored to execute the passive lockon notification routine the next time a port locks on to this port. The routine will then terminate, completing the port-to-port dialogue in the passive role and returning processing control to the user subsystem..

4.5.6.3.3 Termination Routine (Active). Execution of this routine indicates that the passive port was successfully notified of the completion status of the dialogue and is completing its dialogue termination logic. The appropriate dialogue count will then be incremented, and the completion status is passed back to the user activity that initiated the port-to-port dialogue. This terminates IPC until the next dialogue request is received.

4.6 XXBUS Linking Port. For applications requiring more ports than can be accommodated by a single XXBUS, a method of linking multiple XXBUSes is required. This section describes the XXBUS Linking Port (XXBLP) which serves as a hardware component of the inter XXBUS link. The software component for these links is provided by the write only IPC that resides in every processing port.

4.6.1 General Description. The XXBUS Linking Ports (XXBLP) will provide the hardware level links between XXBUSes of the XXBUS network (MPC). Each link will consist of two XXBLPs and a cable connecting the two. Each link is full duplex with data transfer rates comparable to the bandwidth of the XXBUS (80 megabits per second).

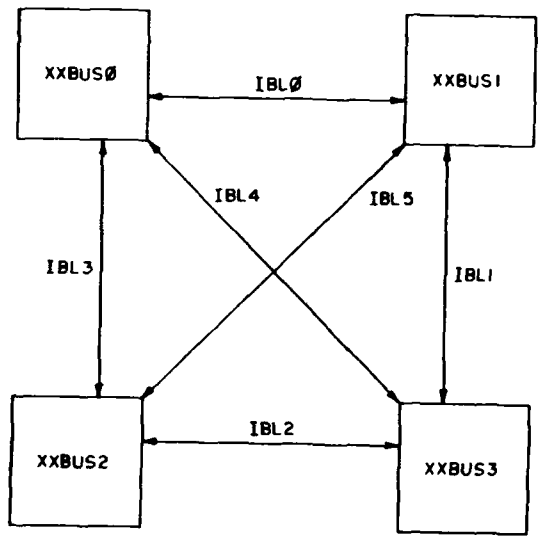
Information is transferred between XXBUSes by these linkes via Information Packes (IP). Each packet corresponds physically to the amount of information written onto a XXBUS during a single bus cycle and logically to the smallest amount of information required to control communications between any two ports in the XXBUS network. This packaging permits the total decoupling of the linked XXBUSes; i.e., bus cycle activity on each bus is completely asynchronous. To further decouple XXBUSes, each XXBLP will contain an IP queue organized as a First-In-First-Out (FIFO) pipeline.

Principle characteristics of the XXBUS linking port design include:

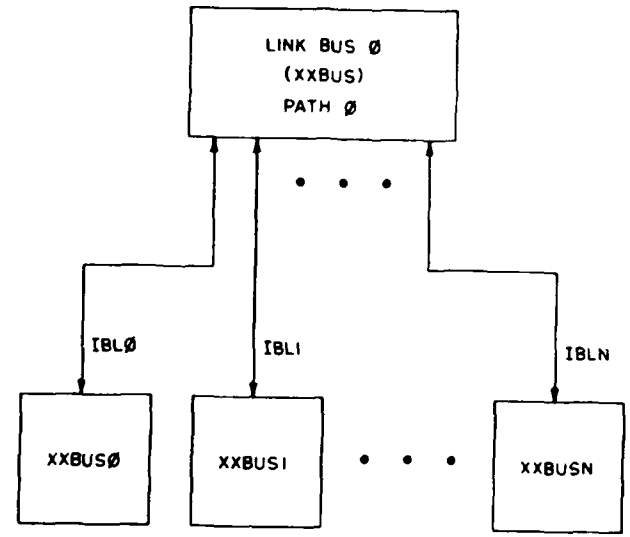
- Bus cycle activity of each XXBUS is asynchronous with respect to one another.

- The Information Packet addressing structure and the XXBLP can support such networking schemes as: Direct only, Bused only, Limited indirect, and combinations of each of these. Figure IV-26 depicts these different shcemes.

- Up to four redundant links or paths can connect any two XXBUSes.

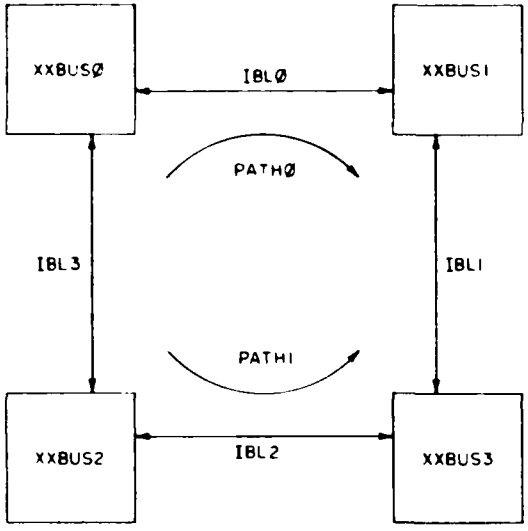


DIRECT NETWORK; NO REDUNDANCY

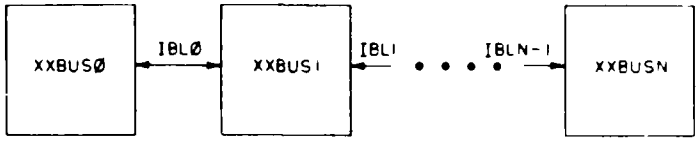


BUSED NETWORK; NO REDUNDANCY

Figure IV-26. XXBUS Network Configurations (1 of 3)



INDIRECT RING NETWORK; TWO RENDUNDANT PATHS



INDIRECT CHAINED NETWORK; NO REDUNDANCY

Figure IV-26. XXBUS Network Configurations (2 of 3)

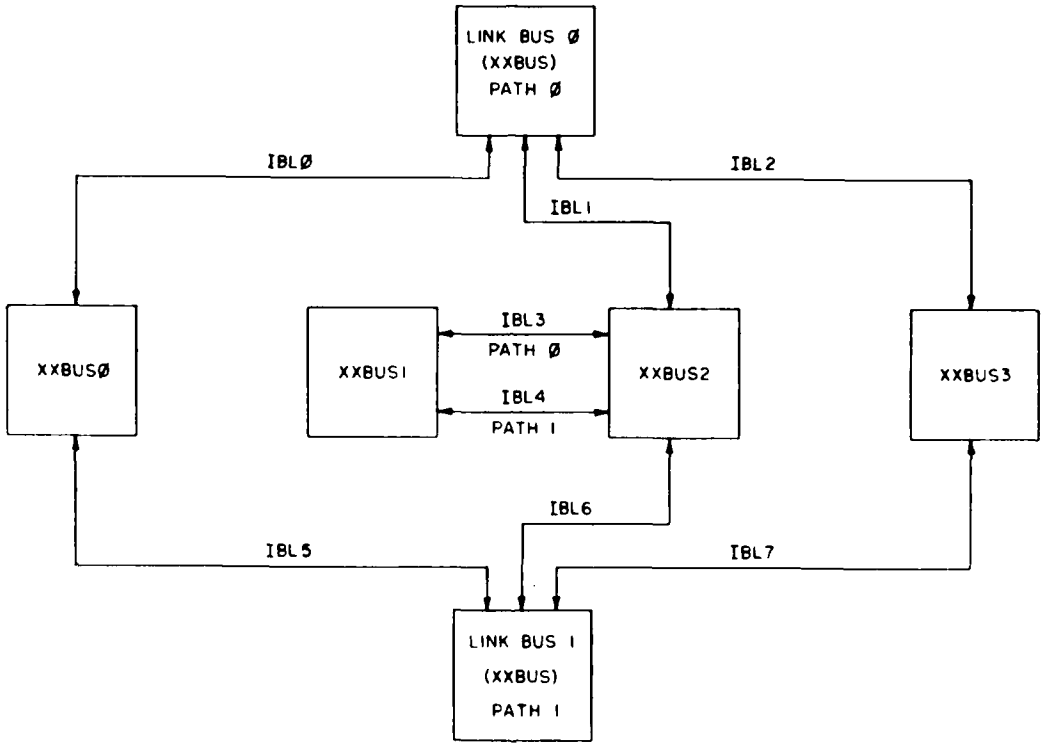


Figure IV-26. XXBUS Network Configurations (3 of 3)

- o Each link is full duplex with data transfer rates comparable to the XXBUS bandwidth.

- o At the bus cycle level the XXBLP appears as any other port.

4.6.2 XXBUS Linking Port Structure. Figure IV-27 shows the functional block diagram of the XXBUS Linking Port (XXBLP). The elements shown are grouped into four units: the XXBUS transmitter, the XXBUS receiver, the IBL cable electronics, and the IP queue. The following paragraphs describe the elements of these units and how they are organized to support the XXBUS linking port's principle function.

4.6.2.1 XXBUS Receiver. The XXBUS receiver of the XXBLP will consist of the XXBUS Information Input Register (XXIR), the Address Recognition Unit (ARU), and the XXBUS receiver logic. Also associated with the receiver will be the following control and status lines: XBCA, XBACK, XBIRL, XBRESET, IKNACK, INRQ, QFULL, XXIRC, and SELECT. See Table IV-02 for a detailed description of each of these signals.

4.6.2.1.1 XXBUS Information Input Register (XXIR). The XXIR will be a 37-bit register with its inputs connected to the XXBUS information but via the XXBUS transceivers and with its outputs connected to the cable drivers. The XXIR will be used to hold IPs that are accepted from the XXBUS by the XXBUS receiver and also serve to decouple the IP queue from the XXBUS.

4.6.2.1.2 Address Recognition Unit (ARU). The ARU will perform the function of comparing address fields of the XXBUS information bus during the early phase of every bus cycle with the path and bus I.D. headers contained within the ARU. This function will determine whether or not the receiver should be selected to participate in the remainder of the bus cycle or not.

The ARUs of the XXBLP differ from those of the processing ports in that the XXBLP ARU will consist of two bus I.D. headers. These two bus IDs will specify the upper and lower limits of the bus ID. The ARU can be configured

SIGNAL	SOURCE	DESTINATION	DISCRIPTION
INACK	IP QUEUE	XXBUS RECEIVER	IN ACKNOWLEDGE. Used by IP queue to indicate to XXBUS receiver that an IP was loaded into the FIFO as requested.
INRQ	XXBUS RECEIVER	IP QUEUE	IN REQUEST. Used by XXBUS receiver to request the loading of the IP contained in the XXIR into the IP queue.
INIP	FIFO CNTRL	FMDB BUFFER	IN INFORMATION PACKET. Used by FIFO control logic to gate IPs from IBL electronics onto the FIFO memory data bus during IP transfers to FIFO memory.
OUTEN	FIFO CNTRL	XXBUS XCVRS	OUT ENABLE. Used by XXBUS transmitter to gate XXOR onto the XXBUS informtion bus during BUS cycle.
QFULL	FIFO CNTRL	XXBUS RECEIVER	IP QUEUE FULL. Used by IP queue to indicate to XXBUS receiver that FIFO is full and will result in the XXBUS receiver operating in the XXIR locked mode.
READ	FIFO CNTRL	FIFO MEMORY	READ MEMORY. Used by FIFO control logic to output the addressed IP onto the FMDB.
SELECT	ARU	XXBUS RECEIVER LOGIC	Used by XXBUS receiver to de termine when to accept a bus cycle.
XBACK	XXBUS RECEIVERS	XXBUS TRANSMITTERS	Bused XXBUS control signal used by XXBUS receiver accepting bus cycle to indicate to XXBUS transmitter controlling bus cycle that the IP was loaded into its XXIR.

Table IV-02. XXBUS Linking Port Signal Definitions (Page 1 of 2)

SIGNAL	SOURCE	DESTINATION	DISCRIPTION
XBCA	XXBUS XMITTERS	XXBUS RECEIVERS	Bused XXBUS control signal used by XXBUS transmitters to indicate to XXBUS receivers and XXBUS control port that a bus cycle has started and that an IP is on the XXBUS information bus.
XBIRL	XXBUS RECEIVERS	XXBUS TRANSMITTERS	Bused XXBUS control signal used by XXBUS receiver accepting bus cycle to indicate to XXBUS transmitter controlling bus cycle that its XXIR is locked and that the IP was not loaded.
XBRESET	XXBUS CONTROL	ALL PORTS	Bused XXBUS control signal used by XXBCP to aid Port initialization during Power Up and System Reset.
XMITRDY	XXBUS TRANSMITTER	FIFO CONTROLLER	TRANSMITTER READY. Used by XXBUS transmitter to indicate to the FIFO controller that the XXBUS transmitter is idle and that the XXOR is available for the next IP.
XMITSTART	FIFO CNTRL	XXBUS TRANSMITTER	TRANSMITTER START. Used by the FIFO controller to start the XXBUS transmitter.
XXBUSGT	XXBCP	ONE FOR EACH XXBUS XMITTER	XXBUS CONTROL GRANT. Used by XXBCP to transfer control of XXBUS to requesting XXBUS transmitter.
XXBUSRQ	ONE FOR EACH XXBUS TRANSMITTER	XBCP	XXBUS CONTROL REQUEST. Used by XXBUS transmitter to request control of XXBUS from XXBCP.
XXIRC	XXBUS RECEIVER	XXIR	Control line used to strobe IP into XXIR.
XXORC	FIFO CNTRL	XXOR	Control line used to strobe IP into XXIR.

Table IV-02. XXBUS Linking Port Signal Definitions (Page 2 of 2)

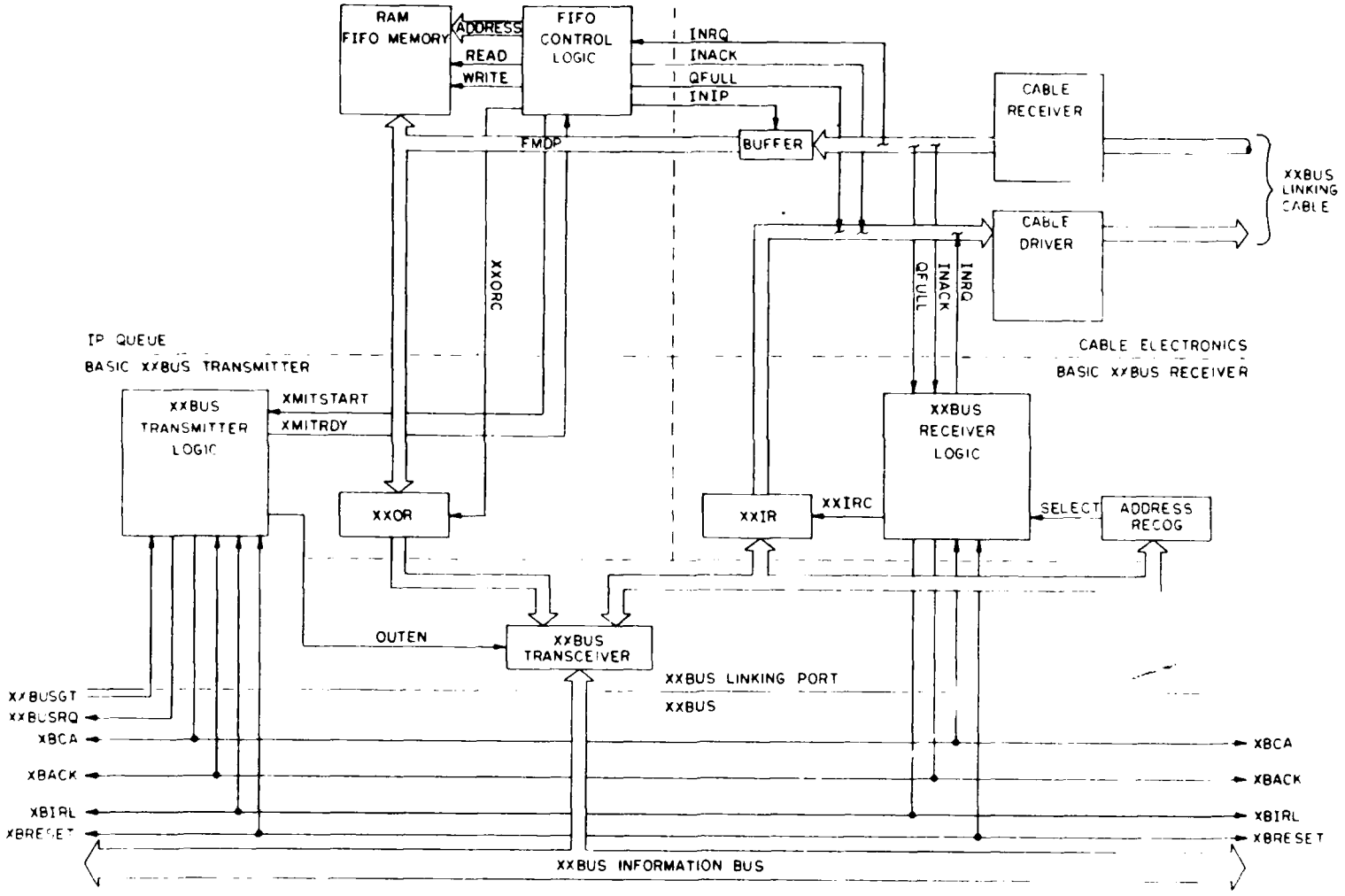


Figure IV-27. XXBUS Linking Port Functional Block Diagram

to accept a bus cycle if the bus ID of an information packet is either inside or outside this range.

Figure IV-28 shows the functional block diagram of the XXBLP ARU. As shown, there are three sets of comparators and ID headers: one for the path ID, and for both the upper and lower bus ID. The outputs of these comparators are connected to a logic network which can be configured, via the setting of the in/out range switch, to make the control line SELECT active conditioned to the IP bus ID being either inside or outside the range specified by the upper and lower bus ID headers. For an example of how the ARU of a network can be figured, see Table IV-03 which shows the configurations of the ARU for the network shown in Figure IV-28.

4.6.2.1.3 XXBUS Receiver Logic. The XXBUS receiver logic will control the transfer of IPs into the XXIR and from the XXIR to the IP queue of the other XXBLP forming the link. This control logic is implemented as an Alogirithimic State Machine (ASM) which must communicate with the transmitter logic of other ports. This communications is accomplished by using the control lines XBCA, XBACK, and XBIRL and the IP queue controller of the linked XXBLP using control lines INRQ, INACK, and QFULL.

4.6.2.2 XXBLP XXBUS Transmitter Elements. The XXBLP XXBUS transmitter elements include the XXBUS information packet output handling register (XXOR) and the XXBUS transmitter logic. Also associated with XXBUS transmitter are the control lines XBCA, XBACK, XBIRL, XBRESET, XXBUSRQ, XXBUSGT, XMITRDY, XMITSTART, XXORC, and OUTEN. Table IV-03 provides a detailed description of each of these signals.

4.6.2.2.1 XXBUS Information Packet Output Register ((XXOR). The XXOR will be used to hold the next IP that is to be transferred from a linking port and will serve to decouple the IP queue from the XXBUS to which the XXBUS transmitter is connected. The XXOR will be a 37-bit register with its input connected to the IP queue and with its output connected to the XXBUS information bus via the XXBUS transceivers.

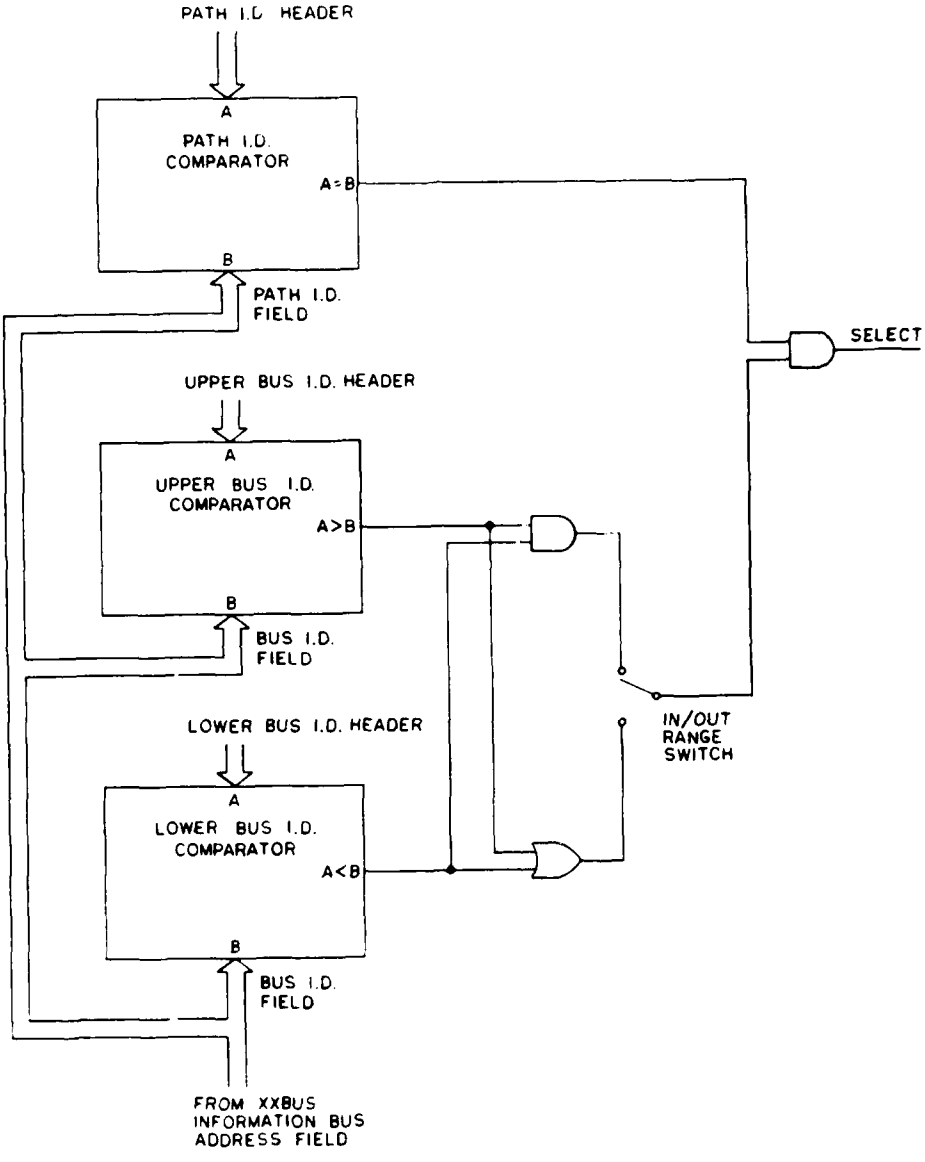


Figure IV-28. XXBUS XXBLP ARU Functional Block Diagram

XXBLP LOCATION		PATH I.D.	LOWER BUS I.D.	UPPER BUS I.D.	IN/OUT RANGE SWITCH
IBL	BUS				
0	XXBUS 0	0	0	0	OUT
0	LBUS 0	0	1	3	OUT
1	XXBUS 2	0	1	2	OUT
1	LBUS 0	0	0	3	IN
2	XXBUS 3	0	3	3	OUT
2	LBUS	0	0	2	OUT
3	XXBUS 1	0	1	1	OUT
3	XXBUS 2	0	0	2	IN
4	XXBUS 1	1	1	1	OUT
4	XXBUS 2	1	0	2	IN
5	XXBUS 0	1	0	0	OUT
5	LBUS 1	1	1	3	OUT
6	XXBUS 2	1	1	2	OUT
6	LBUS 0	1	0	3	IN
7	XXBUS 3	1	3	3	OUT
7	LBUS 0	1	0	2	OUT

Table IV-03. ARU Configuration Table for Network in Figure IV-28

4.6.2.2.2 XXBUS Transmitter Logic. The transmitter logic will control the transfer of IPs from the XXOR to the XXBUS and the transfer of IPs from the IP queue to the XXOR. This control logic will be implemented as an ASM which must coordinate its activities with the IP queue, the receivers of other ports on this XXBUS and with the XXBUS control port. This coordination will be accomplished by using the control lines XMIRDY, XMITSTART, XBCA, XBACK, XBIRL, and the controls XXBUSGT and XXBUSRQ.

The XXBLP XXBUS transmitter logic is similar to that used by the XXBIU. Th only difference is that the XXBLP XXBUS transmitter logic contains a timer used to time out attempts to transfer IP to ports which fail to unlock their XXIRs. For XXBUS transmitters of XXBIU, the time out interrupts and associated microprograms will reset the transmitter if this lock up condition should occur. For the XXBLP transmitters, since XXBLP contains no software component, this time out function must be an integral part of the XXBUS transmitter logic.

4.6.2.3 IP Queue Elements. The IP Queue will provide temporary storage of IPs as they are transferred between linked XXBUSes. It will also enhance the efficiency of this link by increasing the probability that the XXIR of the XXBUS receiver will be unlocked.

The IP queue will be organized as a FIFO pipeline which can accomodate approximately 2000 IPs. The FIFO memory will consist of single port bipolar RAM with read/write access times of less than 60 NSEC. The FIFO control logic will consist of memory address pointers, comparators, and other logic needed to implement the RAM as a FIFO memory. It will also control the transfer of IPs from Inter-Bus Link (IBL) cable to the FIFO RAM and from the FIFO RAM to the XXOR. The FIFO control logic operations will be prioritized to favor transfers from the XXIR to the FIFO RAM. This is done to ensure that the XXBUS receiver will be available for each bus cycle.

The definitions of signals associated with the IP queue are given in Table IV-02.

4.6.2.4 The Interbus Link Cable Electronics. The IBL Cable Electronics include cable drivers, receivers, and the actual cables. The cable electronics will provide a full duplex link between the two XXBLP forming the IBL and will be capable of supporting the 5 MEGA IP transfer rate. The physical nature of the cable electronics will be a function of the environment of the IBL including distance, noise levels, and security requirements. It will also be a function of the types of components that will be available at the time of implementation.

4.6.3 XXBUS Linking Port Operations. The operations of the XXBLP of an IBL will involve the transfer of IPs between the XXBUS which they connect. These operations include determining when to accept IPs from the XXBUS; loading IPs into the XXIR; transferring IPs across the IBL cable to the IP queue of the companion XXBLP; transferring the next IP in the IP queue to the XXOR; and the transferring of IPs from the XXOR across the connected XXBUS to the final destination or to the XXBLP of another IBL. This subsection describes each of these operations with respect to the major functional units of the XXBLP.

4.6.3.1 XXBLP XXBUS Receiver Operations. The Exchange Bus Linking Port (XXBLP) XXBUS receiver will perform the operations involving the transfer of Information Packets (IP) from the XXBUS across the Inter-Bus Link (IBL) to the IP queue of its companion XXBLP. These operations include determining when to accept bus cycles from the XXBUS, transferring IPs from the XXBUS to the XXIR, and transferring IPs from the XXIR to the IP queue. The control of these operations will reside within the XXBUS receiver logic which coordinates its activities with other transmitters on the XXBUS and the IP queue. This coordination will be done via the control lines XBCA/XBACK/XBIRL/INRQ/INACK/QFULL respectfully. A description of each of these signals is given in Table IV-02.

The operation of determining when to accept a bus cycle begins when XBCA is made active. When this occurs the XXBUS receiver will test the state of the select status line from the ARU. The ARU will make SELECT active if the following conditions are met:

- The path ID of the IP must equal that specified by the path ID header.

- If the in/out range switch is in the in position, then the bus ID of the IP must lie inside the range specified by the upper and lower bus ID headers.

- If the in/out range switch is in the out position, then the bus ID of the IP must lie outside the range specified by the upper and lower bus ID headers.

When the receiver tests the control line SELECT, and if SELECT is active, then the XXBUS receiver logic will accept the bus cycle and proceed to engage the transmitter. The bus cycle will proceed in one of two possible ways depending on the state of the QFULL status line. If QFULL is active, then the receiver will operate in the XXIR LOCKED mode. If QFULL is not active, then the receiver will operate in the XXIR UNLOCKED mode. Section 4.4.3.6 describes in detail the operation of the XXBUS receiver in each case.

If, at the completion of a bus cycle, QFULL is not active, the XXBUS receiver will proceed with transferring the IP contained in the XXIR across the IBL cable to the IP Queue. This transfer operation is accomplished with the control lines INRQ and INACK. INRQ is used by the receiver to request the loading of the IP into the IP queue. INACK is used by the IP queue to indicate to the receiver when an IP has been loaded into the IP queue. After the IP is loaded into the IP queue the receiver will become available for the next bus cycle. Since proper operation of the XXBUS transmitters and receivers requires that all receivers on the XXBUS be available prior to the start of a bus cycle, the operation of transfering the IP from the XXIR to the IP queue must have top priority over all other IP queue operations.

If QFULL is active upon completion of the bus cycle the receiver will wait for QFULL to become inactive before starting the XXIR-to-IP queue transfer.

4.6.3.2 Information Packet Queue Operations. The IP queue controller will perform the operations of writing the IP received from the XXBUS receiver of the companion XXBLP into the RAM. It will also track the next in and next out addresses of the RAM, determine when the RAM is full and when it is empty and transfer IP from RAM to the XXBUS transmitter. The control of these operations lies with the FIFO control logic which coordinates these operations with the XXBUS receiver and the XXBUS transmitter. This coordination will be done by the control lines INRQ, INACK, and QFULL, and XMITSTART and XMITRDY. Definitions for these control lines are given in Table IV-02.

The operation of writing an IP into the FIFO RAM will begin with the XXBUS receiver making INRQ active. When INRQ becomes active the FIFO controller will proceed to activate INIP which will gate the IP onto the FIFO Memory Data Buse (FMDB) followed by a write strobe to the RAM. The FIFO controler will at the same time activate INACK which will result in the XXBUS receiver becoming available for the next bus cycle. After the IP has been written into the RAM the FIFO controler will increment its next address counter and check for the queue full condition. If the FIFO is full, then the FIFO controller will set QFULL active which will cause the XXBUS receiver to operate in the XXIR locked mode.

The operation of reading an IP from the RAM and transferring it to the XXBUS transmitter will begin with the existance of an IP in the RAM. If the RAM is not empty, indicating an IP is waiting to be transferred, the FIFO controller will test the state of the XMITRDY status line. If XMITRDY is active then the transmitter will be idle and ready to be loaded with the next IP to be transferred. The IP will then be loaded into the XXOR of the transmitter by the FIFO controller using the strobe line XXORC. After the IP is loaded into the XXOR, the FIFO controller will start the transmitter by activating the XMITSTART control line. At this point, if QFULL is set, the FIFO controller will reset it. Ths operation will be repeated until the memory is empty and can only be interrupted by the transfer of IPs from the XXBUS receiver to the IP queue.

4.6.3.3 XXBLP XXBUS Transmitter Operations. The XXBLP XXBUS transmitter will perform XXBUS acquisition and transferring of IP from the XXOR across the XXBUS to the next destination port. The control of these operations resides with the XXBUS transmitter logic which coordinates these activities with those of the IP queue, the XXBUS receiver of other port on the XXBUS, and with the XXBUS Control Port (XXBCP). This coordination will be done via the control lines XMITSTART and XMITRDY, XBCA, XBACK, XBIRL, and the control lines XXBUSRQ and XXBUSGT. A description of each of these signals is provided in Table IV-02.

When the transmitter is ready to transfer an IP, it will make XMITRDY active to indicate this readiness. When the IP queue has an IP to be transferred and sees XMITRDY active, it will load the IP into the XXOR and proceed to start the XXBUS transmitter by activating XMITSTART. When XMITSTART becomes active, the transmitter will leave its ready state and enter its XXBUS request state. This action will result in XMITRDY becoming inactive, XXBUSRQ becoming active, and in the starting of the transmitter retry time-out timer. The transmitter will then proceed to operate as described in Paragraph 4.4.3.5. At the end of the bus cycle, if the XXIR was unlocked, then the transmitter will return to its ready state; if the XXIR was locked, then the transmitter will return to its XXBUS request state and continue to retry until it is successful or until the retry timer times it out.

SECTION V
SUMMARY OF DESIGN

5. Summary. The Micro Programmable Controller (MPC) is an innovative combination of microcomputer hardware and software synergistically coupled to produce a totally distributed, stand alone computer system. Through the use of parallel processing techniques, large integrated problems can be resolved with the MPC by functionally decomposing the problem and dedicating separate processing resources to each decomposed element. Within this distributed parallel architecture, the requirement for interprocess communications is imperative to ensure coordination and integrity. The previous sections have described design changes to the MPC architecture that will decrease the time to perform interprocess communications. Section 5.1 briefly reviews the components of interprocess communications as they were presented in Section 2. Section 5.2 discusses the major considerations of the improved MPC design as they relate to the components of interprocess communications and provides a table which contrasts current MPC performance characteristics with the goals to be realized with the implementation of the new design.

5.1 Interprocess Communications. Interprocess communications consists of two components: data communications and process coordination. In the Improved Microprocessor Design, the bandwidth of these two components has been increased thereby increasing overall performance of the MPC. The following paragraphs describe various design changes that will increase the bandwidth of each of these interprocess communication components.

Data communication is a measurement of the amount of data that can be transferred between two ports during a given period of time. Data communication bandwidth can therefore be analyzed by determining the raw bandwidth of the XBUS (bus bandwidth) and the effective rate at which two ports can exchange data (port-to-port bandwidth).

- o Bus bandwidth is a measure of bus cycle rate and the number of bits of data transferred per bus cycle. (Cycle rate X number of bits transferred per cycle = bus bandwidth)

- o Port-to-port bandwidth is a measure of port-to-port transfer rate and the number of bits of data transferred per cycle. (Port-to-bus transfer rate X number of bits transferred = effective port-to-port bandwidth)

Process coordination bandwidth is a function of the number of control dialogues a port can perform in a given period of time. Process coordination bandwidth can therefore be determined as the reciprocal of the time required for one control dialogue. This figure will represent the number of control dialogues a port may engage in during the period of one second. (1/total time for a control dialogue = process coordination bandwidth per port per second)

5.2 Improved Microprocessor Design. This design addresses the performance limitations inherent in any distributed parallel architecture - interprocess communications. Interprocess communication is required to ensure the coordination and integrity of parallel resources in a distributed architecture. The time required to perform interprocess communication will ultimately reduce the overall performance of a parallel processing system. Section 4 presented several ways to improve the current MPC architecture by decreasing the time required to perform interprocess communications. The realization of these architectural improvements will significantly increase the system performance of the MPC by achieving interprocess communications performance for single processes comparable with today's serial mainframe systems while providing for a relatively unrestricted number of simultaneous processes. The following paragraphs delineate each of the major MPC design improvements considered during the Improved Microprocessor Design contract. Table V-01, Improved Hardware Performance Characteristics, presents a comparison of the currently operational MPC with the new design.

	CURRENT HARDWARE	IMPROVED HARDWARE
• XBUS Transfer Rate	1 Million CPS	5 Million CPS
• XBUS Data Width	16 Bits Parallel/Cycle	Same
• XBUS Bandwidth	16 Megabit/Sec	80 Megabit/Sec
• Port to Port Bandwidth	200K Bit/Sec	10 Megabit/Sec
• Port Instruction Rate	400K Instructions/Sec	1 Meg Instruction/Sec
• Port Max I/O Rate	1 Megabit/Sec	32 Megabit/Sec (16 Bits)
• Maximum Number of Ports	24 per XBUS 72 per extended XBUS	24-64 per XXBUS 4096 per XXBUS Network
• Maximum Number of Systems	Logically Unlimited via IBC	Same
• Port to Port Interconnect (Process Coord. Bandwidth)	400 per Port per Second	4000 per Port per Second

Table V-01. Improved Hardware Performance Characteristics

5.2.1 Exchange Bus (XXBUS) Improvements. The most significant design change in the XXBUS will be the demultiplexing of the data and address lines. This improvement will increase XXBUS bandwidth by allowing the concurrent transfer of 16 bits of data and a 14 bit address (2 bits for path, 6 bits for bus, and 6 bits for port identification) using a single phase bus cycle. In the current architecture a two phase bus cycle is required to transfer 16 bits of data because the address lines are multiplexed through the data lines. The improved XXBUS will therefore use half as many bus cycles to accomplish this transfer.

5.2.2 Exchange Bus Control Port (XXBCP) Improvements. The most significant design change in the XXBCP will be the simplification of bus cycle arbitration logic. This improvement will increase XXBUS bandwidth by allowing the maximum bus cycle rate to increase to 5 million cycles per second. Bus cycle duration will therefore decrease to a minimum of 200 nanoseconds per cycle. XXBUS bandwidth will thereby increase to 80 megabits per second which is five times greater than the current XBUS bandwidth. The improved arbitration will also remove the priority demand scheme from the request/grant process and service all bus cycle requests equally. This will allow the XXBUS to sustain loading where all of the connected ports are contending for bus cycles without serious degradation to any individual port-to-port bandwidth. In the improved design, an XXBCP will reside on each individual XXBUS therefore, the addition of several XXBUS's into a network will not limit the performance of any single XXBUS or the network as a whole.

5.2.3 Exchange Bus Interface Hardware (XXBIU) Improvements. The most significant design change in the XXBIU will be the decoupling of the Primary Port Processor Unit (PPPU) from the XXBUS. This improvement will increase the port-to-port and process coordination bandwidth of each MPC port by releasing the PPPU from the overhead processing required to initiate and sustain dialogues. Port-to-port bandwidth will increase to 10 megabits per second using DMA transfers across the XXBUS, a 50 fold increase over the current MPC. Process coordination bandwidth will increase to allow 4,000 control dialogues per port per second. The effect of expanding these

bandwidths will ultimately allow each MPC port, and collectively the MPC, to address larger and more complex problems by providing more processing power directed toward to its individual requirements.

The improved XXBIU will preserve the basic architectural characteristics of the MPC while reducing the processes required to access the XXBUS and transfer data to a lower level thereby applying hardware resources in a more efficient manner. The redesign of the XXBIU will require many changes to the Inter-Port Communications firmware subsystem in order to maintain consistency with the improved MPC hardware architecture. It should be noted, however, that redesign of the XXBIU and IPC will not necessitate the redesign of any other MPC subsystem such as ECM, EDR, or MACE.

5.2.4 Inter-Port Communications (IPC) Subsystem Improvements. The most significant design change to IPC will be the separation of IPC into two logical levels. IPC Level 1 will be responsible for the control and execution of all XXBUS data transfer operations and will reside in the Microprogram memory of the XXBIU in each MPC port. IPC Level 2 provides the interface between other MPC software subsystems and IPC Level 1. IPC Level 2 will reside in the ROM of each port and be executed by the port processor. This improvement will consolidate design changes in the XXBUS and XXBIU. IPC will provide port-to-port communications in an environment totally asynchronous to the PPPU at the bandwidths mentioned in the previous sections. The result of IPC improvements, will preserve the proven functionality of the current IPC, while providing a faster more efficient communications interface.

5.2.5 Inter-Bus Linking Port (XXBLP). The design of the XXBLP will allow large networks of MPC XXBUS"S to be configured to resolve problems that require more than 24 (minimum) to 64 (maximum) MPC ports per XXBUS, depending upon MPC cabinetry. The most significant aspect of the XXBLP design will be the asynchronous connection of XXBUS's at the bus cycle level. The XXBLP will operate at the 80 megabit bandwidth of the XXBUS and support networking growth to allow asynchronous interconnection of up to 64 XXBUS's. Such a network would offer 4096 MPC ports to allocate for resolution of system problems. In addition, the XXBLP will support multiple routing between

XXBUS's using the 2 bits of path identification in the 14 bit address. This will provide the system planner with a powerful tool to prevent possible bottleneck that can occur in large, integrated networks.

APPENDIX A

Terms and Abbreviations

ACK/NAK	Acknowledge/Not Acknowledge (error).
ACK/NAK Response	Two bytes displayed by Passive IPC in the ODHR in response to a Transfer Termination Request received from Active IPC.
ACTEN	ACTIVE ENABLE. Mode control line of microprogram control module of XXBIU used to enable the active channel of the XXBIU.
Active Completion Routine	Code called by the Active Direct or Active Indirect Driver when Active IPC returns to EDR for a processing break between DCWs.
Active Direct Service	A routine which is directed towards one specific port and in which the EDR Port plays the active direct role in any dialogue. An Active Direct Service is executed as a result of a dispatch bit being set in the Port Configuration Table.
Active Indirect Service	A routine oriented towards one specific port and executed by EDR in the active indirect mode as a result of an indirect request from the specific port.
Active Port	(1) An MPC Port that contains the hardware necessary to request exchange bus usage. (2) The port which drives an IPC dialogue by sending interrupts to the passive port.

ACTRDY	ACTIVE READY. Interrupt request line connecting XXBIU to PPPU interrupt handling hardware.
DCCP	Advanced Data Communications Control Protocol
ADMAP	Active DMA Address Pointer. Located in DMA module of XXBIU. Used during active DMA transfer to point to next word to be transferred.
ADMAWC	Active DMA Word Counter. Located in DMA module of XXBIU. Used to determine end of active DMA block transfer.
APMPAR	Active Port Microprogram Address Register. Register of Address Queue.
APPR	Active Page Pointer Register. Points to the page that contains a block to be transferred during active DMA.
AQ	Microprogram Address Queue. Registers located in microprogram module used for addressing the microprogram memory.
ARC	Active Redundant Check word generator. Located in XXBUS module of XXBIU. Used to generate check word during DMA operations.
ARU	Address Recognition Unit. The element of each XXBUS receiver which determines if the XXBUS receiver should engage a bus cycle.

ASM	Algorithmic State Machine. A type of sequential state machine used to implement hardware control logic.
ATC	Active DMA Terminal Count. Connects a microprogram module with a DMA module. Used to indicate the end of an active DMA block transfer.
ATO	Active Time Out. Control line connecting the microprogram control module to the dialogue timer.
Attention Bit	(Curent MPC). One of the status bits which is used to synchronize the active and passive ports in a dialogue.
Availbility (or Passive Availabilty)	The passive communication modes and passive service groups which some channel within a port has requested.
AXMPAR	Active XXBUS Microgram Address Register. Register of Address Queue.
BAR	Current Exchange Bus Address Register.
BIM (or BIMA)	(Current MPC). Exchange Bus Interface Module. The BIM grants a bus cycle to a requesting port through the XIM in that port's cabinet.
BMMB	Branch Microprogram Module Bus.

BR-90	Bunker Ramo Graphics Console.
BUS ID	6-Bit field of XXBUS address used to specify Bus of destination Bus.
CCT	Channel Control Table. IPC Table.
CCU	Channel Control Units. Used to interface BR-90 (PRE-MPC) with PACER
Channel	A long-term interface between IPC and the user-level program.
CLU	Conditional Logic Unit. Located in XXBUS module of XXBIU. Used to decode data control words.
CMPR1	Compare Register One. Located in XXBUS module of XXBIU. Used with CLU operations.
CMPR2	Compare Register Two. Same as CMPR1.
COMCND	Compare Condition. Status line connecting XXBUS module to microprogram control module. Used with CLU operations.
Completion Routine	A routine called by Passive IPC at the end of a header or segment transfer, or at the end of a dialogue.
Control Word	The last word in a DCW which provides information concerning the data block described by the DCW.

Control Interrupt	(Current MPC). A command written across the bus at the interrupt seven level.
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check.
CS	Chip Select.
Data Block	A contiguous block of memory to be transferred by IPC.
Data Interrupt	(Current MPC). A command transferred across the XBUS at the interrupt five level.
DCW	Dialogue Control Word.
DDCMP	Digital Data Communications Message Protocol.
DDE	Data Decoding Elements. Elements of XXBUS module that support XXBUS data decoding.
DEC	Digital Equipment Corporation.
DECX	Decode and Execute Function Code. Control line connecting queue controller with XXBUS command unit of the MCM.
Dialogue	An interaction between IPC in two ports which establishes a connection, transfers the desired data and directives, and terminates the connection.

Dialogue Termination Code	See Termination code.
Dialogue Termination Request	Two bytes written by Active IPC to the connected passive port when a dialogue is successfully completed or when a dialogue is terminated due to error.
Direct Communication	An inter-processor communication in which IPC in the requesting processor is granted bus usage and sends a message to a desired port.
DMA	Direct Memory Access
DMAM	Direct Memory Access Module. Located in XXBIU. Contains address registers and control logic that supports DMA between the port memory and the XXBUS.
DMAMCU	DMA Module Control Unit. Control logic that executes DMA module MI directives.
DOD	Department of Defense.
DXEX	DMA and XXBUS Modules Execute. Control line connecting XXBUS, DMA, and microprogram control module. Used to tell XXBUS and DMA module to execute MI directive.
DXRDY	DMA and XXBUS Modules Ready. Control line connecting DMA, XXBUS, and MCM modules of XXBIU. Used to indicate DMA and XXBUS modules ready for next MI directive.

ECM	External Control and Monitoring Software Subsystem.
EDR	Error Detection and Recovery. MPC Software Subsystem.
EMI	Electromagnetic Interference.
EXPARQ	Execute Port Active Request. EXRQ line.
EXPPRQ	Execute Port Passive Request. EXRQ line.
EXRQ	Execute Request. Internal control lines of queue controller used to request execution of microprograms.
EXXARQ	Execute XXBUS Active Request. EXRQ line.
FEP	Front End Processor.
FMS	File Management System.
FMC	File Management Control.
FIFO	First-In, First-Out
FMDB	First-In, First-Out Memory Data Bus. Located in XXBUS linking port.
FUNCTION CODE	Used to specify function of XXBIU to be executed.

GETPB	Get Port Bus. Internal Control line of DMA module.
GRANTED	Control Line Located in XXBUS Control Port. Indicates that arbiter is in granting state.
GROUPRQ	Control Line Located in XXBUS Control Port. Used by port arbiter to make request to group arbiters.
HANDSHAKING	Method of transfering information between asynchronous control structures.
HBGPAP	High Byte of General Purpose Address Pointer. Located in DMA module.
HCR	Header Completion Routine.
HDLC	High Level Data Control Link.
Header	A segment which contains information describing the desired dialogue.
HIS	Honeywell Information Systems.
I/O	Input/Output.
IBC	Inter-Bus Communications. MPC Software Subsystem.
IBL	Inter-Bus Link. Hardware link used to network XXBUS.

IDHR	Old Exchange Bus Input Data Handling Register.
IDHS	Intelligence Data Handling System.
INACK	In IP ACKNOWLEDGE. Control line used to acknowledge INRQ from XXBUS receiver.
Indirect Communication	An inter-processor communication in which the requesting processor enters the request in its ODHR and waits for this request to be read by the desired port.
INRQ	In IP REQUEST. Control line used by XXBUS receiver to request transfer of IP from XXIR.
Internal port	The hardware by which an Intel 8080A CPU communicates with other parts of the microprocessor or port.
Interrupt	A command by which one port reads or writes the bus interface registers of another port and attempts to cause the target port to execute at a location determined by the bus interface hardware.
Interrupt-Level Processing	Program execution caused by the receipt of an interrupt and which causes the user-level program to be temporarily suspended.
IOR	Input/Output Read.

IOW	Input/Output Write.
IP	Information Packet. Basic unit of information exchanged between MPC ports via XXBUS.
IPC	Inter-Port Communication. MPC Software Subsystem.
IPC L1	IPC Level One. IPC microprograms that are executed by XXBIU.
IPC L2	IPC Level Two. IPC residing in firmware of PPPU.
JMP	JUMP. Branch operation.
LBGPAP	Low byte of General Purpose Address Pointer. Located in DMA module of XXBIU. Used to make single word DMA accesses.
LBUS	Linking Bus.
Linking Bus	XXBUS containing only XXBUS Linking Ports. Used to link XXBUSES.
LLLP	Last-Looked-at, Lowest Priority.
LOB	Lock On Request Bit. A bit within the function code used to indicate that function code is lockon request.

Lockon Request	(Current MPC). Two bytes written by Active IPC to a target port requesting a dialogue.
Lockon Response	(Current MPC). Two bytes displayed in the ODHR by Passive IPC to indicate that it has accepted an active port's Lockon Request.
LORQEN	Lock On Request Enable. Enable line used by IPC L1 to indicate willingness to accept lock on request to the XXBIU.
LSB	Least Significant Byte or lower byte of a two-byte grouping; least significant bit.
MACE	MPC Asynchronous Control Element. MPC Software Subsystem.
Machine State	The current state of all registers in a CPU, including the processor word, instruction counter, and stack pointer.
MBR	Module Branch Register. Located in the MCM of the XXBIU. Holds target module address during execution of branch micro-instructions.
MCM	Microprogram Control Module.
MCMCU	Microprogram Control Module, Module Control Unit.

MCU	Module Control Unit. Logic contained within the XXBUS, DMA, and MCM modules that control the fetch and execution of microinstructions.
MDAR	Microprogram Module Address Register. Register of XXBIU MCM address queue.
MEMRD	Memory Read. Control line used by DMA module to read from port memory.
MEMWT	Memory Write. Control line used by DMA module to write to port memory.
MI	Microinstruction.
MICROINSTRUCTION	A single word of the XXBIU microprogram memory containing directives that are executed by the XXBIU hardware.
MICROPROGRAM	A collection of microinstructions which implement a function of the XXBIU. The XXBIU can contain up to 32 microprograms.
Microprogram Control Module	A hardware module of the XXBIU which provides for high level control of the XXBIU and for the control of microprogram execution.
MICROPROGRAM MEMORY	Memory of the XXBIU which contains the microprograms of the XXBIU.

MICROPROGRAM MODULE	A subdivision of a microprogram consisting of up to 8 microinstructions. When the XXBIU executes a module, the module maintains control of the XXBIU until it terminates/subterminates. A microprogram can contain up to 8 modules.
Microsecond(s)	.000001 second.
MI Directive	Microinstruction field reserved for directives of a XXBIU module.
MIEL	Microinstruction Execution Logic. Control logic located in the MCMCU that controls the fetch and execution of microinstructions.
MIELEX	Microinstruction Execution Logic Execute. Control line connecting MCMCU and queue controller of XXBIU MCM.
MIELRDY	Microinstruction Execution Logic Ready. Control line connecting MCMCU and queue controller of XXBIU MCM.
MIHR	Microinstruction Holding Register. Registers associated with the XXBUS, DMA, and MCM module control units. Used to hold microinstruction during execution.
MILU	Microinstruction Loading Unit. Control Logic located in DMA module.
MM	Microprogram Memory.

MP	Microprogram
MPC	MICROPROGRAMABLE Controller.
MPCC	Multiprotocol Communication Controller.
MPMAB	Microprogram Memory Address Bus. Circuit path of XXBIU connecting address queue of MCM to the microprogram memory.
MPMAR	Microprogram Memory Address Registers. Registers of the XXBIU MCM address queue.
MPMDB	Microprogram Memory Data Path. Circuit paths of the XXBIU used to transfer microinstructions from the microprogram memory to the XXBIU modules.
ms	Millisecond (.001 second).
MSB	Most Significant Byte or upper byte of a multiple byte grouping.
Network Direct Communication	A communication mode characterized, in general, by the exchange of information between user-defined channels.
NOP	No Operation. Null program instruction.
ODHR	Old Exchange Bus Output Data Handling Register.
OJ-389(V)/G	Sperry Univac Alpha Numeric/Graphics Console.

OUTGT	OUT GRANT. Control line located in XXBUS control port used by XXBUS cycle sequencer to tell arbitration unit to output queued grant.
PACER	Program Assisted Console Evaluation and Review.
PAS	Passive Active Switch. Control bit of function code used to specify the passive channel of the XXBIU.
PASEN	Passive Enable. Control line of microprogram control module of XXBIU used to enable the active channel of the XXBIU.
PASRQ	Passive Request. Interrupt request line connecting XXBIU to PPPU interrupt handling hardware.
Passive Availability Display	(Current MPC). Two bytes displayed in a port's ODHR when that port is available to enter a dialogue.
Passive Port	(1) An MPC Port that lacks the hardware necessary to request XXBUS usage. (2) The port which is driven through a dialogue by the XXBUS interrupts received from the active port.
Passive Service	A routine executed by EDR in the passive role as a result of a request written to EDR by an external active port.

Path ID	2-Bit field of XXBUS address field used to specify the path that an IP is to take.
PBAU	Port Bus Acquisition Unit. Control unit of the DMAM.
PBAVL	Port Bus Cycle Available. Internal control line of DMA module.
PCOM	PACER Communication Modules used to integrate PACER and MPC.
PCU	Port Command Unit. Located in microprogram control module. Provides command level interface betwen PPPU and XXBIU.
PCT	Port Control Table. IPC table.
PCU	Port Command Unit.
PDMAP	Passive DMA Address Pointer. Located in DMA module of XXBIU. Used during active DMA transfers to point to the word to be transferred.
PDMAWC	Passive DMA Word Counter. Located in DMA module of XXBIU. Used to determine the end of passive DMA block transfer.
PMB	Port Memory Bus.
Port	The physical device containing and executing the various MPC Software Subsystems. Each is a complete microcomputer consisting of Central Processing Unit, memory, bus interface hardware, and (in general) I/O device handlers.

Port Data Bus	Circuit Path Connecting Port Memory to XXBIU.
Port ID	6-Bit field of XXBUS address used to specify destination port of IP.
PORTRQ	PORT REQUEST. Control line of the XXBUS control port's arbitration unit. Used by port arbiter to make request to group arbiter to issue grant.
PPMPAR	Passive Port Microprogram Address Register. Register of Addres Queue.
PPPR	Passive Page Pointer Register. Points to page that contains block to be transferred during passive DMA.
PPPU	Primary Port Processing Unit. The processing elements that perform the principle tasks of the port. Includes the CPU(s), port memory, and other support hardware.
PRCG	Passive Redundant Check word generator. Located in XXBUS module of XXBIU. Used to generate check words during passive DMA operations.
PS $\overline{RQ}/\overline{GT}$	Single, bi-directional handshaking line connecting PPPU and XXBIU. Used to transfer control of Port System Bus.

PTC	Passive DMA Terminal Count. Connects microprogram module with DMA module. Used to indicate end of passive DMA block transfer.
PTO	Passive Time Out. Control line connecting microprogram control module to dialogue timer.
PXMPAR	Passive XXBUS Microprogram Address Register. Register of address queue.
QFULL	QUEUE FULL. Control line of the XXBUS linking port used by IP queue controller to tell XXBUS receiver that IP queue is full.
QUEGT	QUEUE GRANT. Control line of XXBUS control port connecting XXBUS sequencer with arbitration unit. Used to tell arbitration unit to prepare to issue next grant.
RADC	Rome Air Development Center.
RAM	Random Access Memory.
RCA	Radio Corporation of America
RCVE	Receive.
RELPB	Release Port Bus. Internal control line of DMA module.

RNP	Remote Network Processor.
ROM	Read Only memory.
SAC	Strategic Air Command.
SCR	Segment Completion Routine.
SDLC	Synchronous Data Link Control.
Segment	A group of one or more data blocks which is transferred across the bus in one directioon and which is stored in a contiguous memory location in the receiving port. A segment may contain either header information or data.
SELECT	Control line used by address recognition unit to tell XXBUS receiver to engage in bus cycle.
Service Group	(1) A bit or number that indicates a port type. (2) A number of corresponding bit mask used in the active indirect communication mode to request or offer a group of services. EDRs service group numbr is zero, which translates to a bit mask of 80_H.
SIU	Status Interface Unit. Unit of the Microprogram Control Module. Used by PPPU to read status of XXBIU and its microprograms.

SOW	Statement Of Work.
SPLC	Special Purpose Communication Link.
STARTIMER	Control line located in XXBUS control port used by the XXBUS cycle sequencer to time XXBUS cycle.
Status Bits	(Current MPC). Four bits in a port's bus interface hardware indicating if the port is available for a dialogue, if the port is executing, if the I/O drivers are enabled, and if the port is servicing an interrupt.
Status Read/Write	(Current MPC). A command by which one port reads or modifies the current state of a port's status bits.
System Direct Communication Code	A direct communication mode generally used in the execution of control functions and which frequently involves the IPC - maintained channel in the target port.
Terminate Bit	The highest bit in an ACK/NAK (or Dialogue Termination Code.)
Termination Code	A code indicating either the successful termination of a dialogue or the reason for unsuccessful termination. The code consists of a terminate bit and an ACK/NAK code (lowest seven bits).

TIMEOUT	Control line located in XXBUX control port. Used by XXBUS cycle timer to indicate the XXBUS sequencer bus cycle timed out.
Transfer Termination Request	Two bytes written by Active IPC to the connected passive port when a data block transfer is completed.
Visibility	(Current MPC). A display in a port's ODHR which initiates the port's passive availabilty.
XIM	(Current MPC). Exchange bus buffer module. Each MPC cabinet contains a XIM which forwards a port's requst for a bus cycle to the BIM.
XXIDR	XXBUS Input Data Register. Holding register for data received from XXBUS.
XXIFR	XXBUS Input Function Code Register. Holding register for Function Codes received from XXBUS.
XXIR	XXBUS Input Registers. The holding registers XXIFR, XXIDR, and in XXBUS linking ports XXIAR.
XXIRC	Control Line used to strobe data into the XXIR.

XXIRLOCKED	XXBUS Input Registers Locked. Used to indicate to an XXBUS receiver that the XXBUS input registers are locked; if active, an XXBUS receiver will operate in the LOCKED mode.
XXODR	XXBUS Output Data Register. Holding register for data to be transmitted to XXBUS.
XXOFR	XXBUS Output Function Code Register. Holding register for function codes to be transmitted to XXBUS.
XXOR	XXBUS Output Registers. For XXBIU, includes the registers XXODR, XXOFR, and XXAOAR/XXPOAR. For XXBUS linking ports, holding register for Information Packets to be transmitted to the XXBUS.
XXORC	Control line used to strobe data into the XXOR.
XXPOAR	XXBUS Passive Output Address Register. Holding register for passive channel XXBUS address to be transmitted to XXBUS.
XBACK	XXBUS control line used by XXBUS receivers in XXIR UNLOCKED mode to engage XXBUS transmitters. Indicates Information Packet was accepted.
XBCA	XXBUS control line used to indicate when a XXBUS transmitter has started a bus cycle.

XBIRL	XXBUS control line used by XXBUS receivers in XXIR LOCKED mode to engage XXBUS transmitters bus cycle. Indicates Information Packet was not accepted.
XBRESET	XXBUS RESET. XXBUS control line sourced by the XXBUS control port. Used to reset the MPC ports of a XXBUS during power up and system reset.
XBUS	Current MPC Exchange Bus.
XBM	XXBUS Module. Module of XXBIU that performs XXBUS data transmitting, receiving, and error detecting.
XBMCU	XXBUS Module Control Unit. Control logic of XXBUS module that controls the execution of XXBUS module MI directives.
XCU	XXBUS Command Unit. Unit of microprogram control module that provides the XXBUS with command level control of a ports XXBIU.
XMIT	Transmit.
XMITRDY	Transmitter Ready. Used by XXBUS transmitter to indicate that it is ready to transmit the next information packet.
XR	XXBUS Receiver.
XT	XXBUS Transmistter

XTB	XXBIU Transfer Bus. A circuit path of the XXBIU over which the DMA, XXBUS, and MCM module transfer information between each other and port memory data bus.
XXAOAR	XXBUS Active Output Address Register. holding register for active channel XXBUS address to be transmitted to XXBUS.
XXBCP	XXBUS Control Port. Port that controls use of the XXBUS.
XXBIU	XXBUS Interface Unit. A microprogrammable controller that interfaces a ports PPPU with the XXBUS.
XXBLP	XXBUS Linking Port. Port used to link XXBUSES.
XXBUS	Improved MPC Exchange Bus. A common communications channel over which MPC ports communicate.
XXBUSGT	XXBUS Grant. Control lines used by XXBUS control port to grant use of XXBUS to requesting XXBUS transmitter.
XXBUSRQ	XXBUS Request. Control line used by a XXBUS transmitter to request from XXBUS control port use of the XXBUS.
ZIF	Zero Insertion Force. Board connectors built by AMP and used in the MPC cabinet.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

END

DATE
FILMED
3-82

DTIC